

Дмитрий Садовник

Системы машинного зрения

Крым 2026

Оглавление

Оглавление	1
Предисловие.....	5
Об авторе.....	7
Часть 1	12
Хронология становления машинного зрения	12
.....	12
1950-е: Ранние основы	12
1960-е: Формирование задач и первых алгоритмов.....	18
1970-е: Новые методы анализа изображений	23
1980-е: Алгоритмы и нейросети – от теории к практике.....	26
1990-е: Статистические методы, функции и данные	33
Часть 2	50
Распознавание образов	50
Каскадная детекция Виолы–Джонса (Viola-Jones).....	54
HOG.....	63
NMS подавление немаксимумов.....	67
SURF	69
CUDA.....	70
PASCAL VOC	72
ImageNet	75
ILSVRC	77
AlexNet	78
CNN	80
CNN Классификация: когда всему изображению нужна одна или несколько меток	83
CNN детекция объектов: где именно и что именно	88
CNN Сегментация: «раскрасить» сцену по смыслу.....	94
CAFFE	99
MS COCO.....	101
VGGNet	105
GoogLeNet (Inception).....	107

GAN - Генеративные нейронные сети	108
R-CNN	110
ResNet.....	113
SSD	115
YOLO (You Only Look Once).....	118
TPU.....	122
TensorFlow	124
PyTorch.....	126
Распознавание позы	127
OpenPose	132
EfficientNet.....	135
Kinetics-400 и 3D ConvNet.....	138
Трансформеры.....	142
ViT - Vision Transformer	145
Часть 3.	148
Трекинг объектов.....	148
Object Tracking Benchmark (OTB)	150
Трекер MIL	152
Трекер TLD	154
Трекер MEDIANFLOW	156
Трекер MOSSE	158
Трекер KCF	160
Трекер GOTURN.....	162
Трекер CSRT	164
LaSOT: Масштабный бенчмарк для долгосрочного трекинга объектов	166
TrackingNet.....	169
ATOM.....	172
DiMP: «обучаемый оптимизатор» для трекинга (ICCV 2019).....	176
Сиамское семейство трекинга.....	178
MOT трекинг	181
Практическое применение трекеров в бильярде.....	182
RE-ID-трекинг	184
AI City Challenge	187

ByteTrack.....	189
MOTR	192
TimeSformer	196
ЧАСТЬ 4 Автономный транспорт	199
L0 – Без автоматизации.....	202
L1 – Помощь водителю.....	202
L2 – Частичная автоматизация	202
L3 – Условная автоматизация.....	202
L4 – Высокая автоматизация.....	203
L5 – Полная автоматизация	203
Раннее зарождение отрасли	204
Эволюция алгоритмов автономного вождения	211
Отечественная школа автономного вождения.....	215
Рождение эры автономного транспорта.....	219
Часть 5	224
Видео-понимание: видеть не кадр, а историю	224
Видео-свертки и понимание движения: как «слышать» время в кадре.....	229
3D-восприятие: из плоского кадра к миру с глубиной	232
OCR	236
Часть 6	240
Взгляд в будущее	240
Гипотеза внутреннего «пространства качеств» в нейросетях	244
Локальные качественные градиенты: свертки как измерители изменений.....	246
Глобальная интерпретация: внимание и «синтаксис качеств»	249
Резонанс качеств: устойчивые гипотезы и достраивание недостающего	251
Итоги и перспективы: от устойчивых качеств к динамическому атласу мира.....	256
Благодарности.....	265

Предисловие

Эта книга написана для того, чтобы дать вам представление о современных технологиях машинного восприятия и, в частности, машинного зрения, погрузить вас в его технологический контекст. Достижения, которые произошли в этой области за прошедшее десятилетие, просто невероятны. Эта история успехов человеческого разума в отдельно взятой сфере характерна, я уверен, для всех остальных сфер современного научно-технического творчества и наглядно отражает его базовые принципы. Изобретательность, сотрудничество, осознание возможностей, обобщение. Все начинается с технологического контекста времени. Изобретательный разум непрерывно ищет в нем новые возможности. Книга поможет сориентироваться в имеющихся достижениях в машинном зрении и понять как и откуда оно появилось. Пока нет универсального искусственного интеллекта, его применение адаптировано под конечный набор более узких базовых задач. Но общая тенденция по их унификации четко прослеживается на исторической перспективе от момента появления машинного зрения до наших дней. Эту тенденцию я показываю в этой книге. По ней можно понять дальнейшее направление технического прогресса в области искусственного интеллекта и перспективы его применения.

Я, как автор книги, отношу себя к заядлым практикам. Поэтому в книге я собрал множество примеров как из своего личного опыта разработки систем на базе машинного зрения, так и из общеизвестных фактов. Уверен, что это упростит понимание сухой теории и математических моделей. В конечном счете все сводится к вашей личной мотивации изобретать что-то новое. Поэтому личные примеры и пути людей, которые двигаются в общем направлении, имеют большую ценность.

Эту книгу в известном смысле можно считать практическим пособием по решению изобретательских задач. Она предназначена для практиков в области машинного зрения, стремящихся эффективно решать реальные задачи в условиях строгих ограничений — от вычислительных ресурсов и качества данных до динамики сред. Она предлагает всесторонний обзор инженерных подходов, накопленных коллегами за полвека развития дисциплины, — от ранних каскадных классификаторов и локальных дескрипторов до современных трансформеров и мультимодальных систем, — проверенных временем в промышленных и исследовательских проектах. Эти методы, адаптированные к разнообразным сценариям, от автономного вождения до анализа спортивных трансляций, демонстрируют, насколько гибкими и многогранными могут быть ре-

шения: например, один и тот же вызов, такой как трекинг объектов в окклюзиях, решается через простые фильтры Калмана или глобальное внимание, в зависимости от контекста. Зная эту палитру, читатель сможет не изобретать велосипед, а целенаправленно модифицировать проверенные стратегии под новые реалии, ускоряя внедрение и повышая надежность систем.

Об авторе



Когда я решил начать свой путь изобретателя в сфере машинного зрения, у меня уже имелась база технических способностей по программированию и соответствующий опыт работы. Этот опыт был в сфере фулстек клиент-серверного программирования на `nodejs` + `mongodb` или программирования полного цикла.

В юности мой отец приобщил меня к бильярдному спорту, которым я всерьез увлекся. Последующие семь лет я занимался им с большим энтузиазмом

вплоть до второго курса института, когда стало понятно, что лучший в мире бильярдист из меня уже маловероятен, и надо как-то удовлетворять свои амбиции другим способом. Только единицы способны стать лучшими в спортивной карьере. К тому же я родился и вырос на периферии бильярдного движения, которое в основном сосредоточено в офлайн-формате в крупных городах.

В результате моего углубленного знакомства с миром бильярда выкристаллизовалось понимание всей важности наличия высокой духовной составляющей не только в спорте, но и в жизни в целом. После бильярда я направил внимание на духовные поиски навстречу самому себе и своему предназначению. На протяжении последующих еще семи лет я на практике изучал глубинные принципы работы нашего сознания, интересовался глобальным устройством мироздания и человеческим путем развития.

В определенный момент я осознал свое предназначение в деле развития цифровых технологий и почувствовал желание начать приносить конкретную пользу людям через эти технологии. Я погрузился в сферу прикладного программирования, в качестве инженера-разработчика выполнял коммерческие задачи, накапливая опыт ответственного исполнителя. В результате мой бильярдный энтузиазм, накопленный опыт технического творчества и контекст, в котором я тогда работал —

статистика и аналитика больших данных, — собрались в идею собственного проекта — систему бильярдной аналитики на базе машинного зрения.

Хорошо помню этот день как сейчас. Мы гуляли с другом моего детства Сергеем Лохматовым, я рассказывал ему про перспективы развития машинного зрения, про эффективный, на мой взгляд, путь его освоения — выбор и решение прикладной практической задачи. Такой задачей как раз показалась система распознавания и анализа спортивных бильярдных ударов по видео. На тот момент я не знал полностью автоматической системы подобного рода. Но перспективы использования, если бы она появилась, казались интересными в первую очередь со спортивной точки зрения. Это открыло бы возможность объективной оценки спортивного уровня игрока в автоматическом режиме в любое время. Я рассуждал так: когда я наблюдаю за игрой спортсмена высокого уровня, я могу визуально ее оценить с высокой достоверностью и отличить такую игру от аналогичной попытки игрока любительского уровня. Им даже не нужно играть против друг друга непосредственно. Для меня, как для опытного наблюдателя и игрока, видевшего многие матчи, очевидно, чем именно спортивный уровень и подготовка спортсменов отличается. Мне показалась реалистичной гипотеза, что и компьютер сможет это осуществить на базе машинного зрения с последующим блоком аналитики. И если это получится, то бильярдный спорт только выиграет от наличия такого инструмента, и его популярность тоже подрастет. Почему так, стало понятно позднее.

Немного отойду от рассказа о себе. Все виды спорта решают одинаковую задачу — сортировка спортсменов, но делают это разными способами. Пример прямой сортировки — это гонки. Участники стартуют одновременно из установленной точки и мчатся к финишу как можно быстрее. Затем всех участников сортируют по времени прибытия, и это и есть главный объективный показатель их спортивного уровня. Результат сортировки можно узнать сразу, когда к финишу придет последний спортсмен. Даже при их огромном количестве, в несколько тысяч, процесс занимает фиксированное время и дает высокую объективность. Другая ситуация в игровых видах спорта. Здесь, для того чтобы составить рейтинг игроков, нужно провести турнир, на котором спортсмены или команды будут мериться силами напрямую по парам. Проблема в том, что время проведения турнира находится в прямой зависимости от количества участников. А общая достоверность результатов не настолько объективна как в гонках, поскольку часто нет возможности провести матчи каждый между каждым напрямую. Уверенно можно узнать только победителя, остальное распределение мест среди участников имеет меньшую уверенность и сильно зависит от стечения обстоятельств. Проблема частично решается рейтинговыми системами, которые опираются

на накопление очков за результаты участия не в одном турнире, а в целых турнирных сериях и сезонах.

Продвинутая система машинного зрения — альтернативный способ решения этой задачи с выдающейся эффективностью и скоростью, без каких-либо неожиданных сбоев или сюрпризов. На первый взгляд, это может показаться неочевидным, особенно в наше время, когда спорт превратился в полноценную индустрию с миллиардными оборотами, но в его основе лежит фундаментальная потребность человеческой психики и мозга — оценка собственного уровня по сравнению с себе подобными. Это не просто соревнование навыков, а глубокая инстинктивная потребность определить свое место и место других в социальной иерархии, которая пронизывает все аспекты нашей жизни. Мы подсознательно выстраиваем такие иерархии во всем, с чем сталкиваемся: от повседневных взаимодействий до глобальных структур общества. Очевидно, что эта базовая программа эволюционировала как мощный инструмент выживания, помогая нашим предкам координировать группы, распределять ресурсы и строить цивилизации; она передается генетически на протяжении тысяч лет, закаленная в горниле естественного отбора. В природе эта задача играет огромную роль, проявляясь на самых глубоких уровнях творения — от инстинктов животных до сложных человеческих сообществ. В обществе сложились общепризнанные эталоны для градации этой иерархии: физическая сила, красота, скорость реакции, меткость движений, выносливость, здоровье тела, сообразительность и, наконец, интеллектуальная глубина. Эти качества передаются участникам сообщества неявным, бессознательным способом, всегда в контексте эпохи и культуры, формируя невидимые нити социального порядка. Вопрос о приоритетах этих эталонов — сила ли важнее ума, а счастье и свобода ли перевешивают краткосрочную выгоду — стоит оставить для философских размышлений; каждый решает его сам, исходя из своей жизненной ситуации, перспектив и того, откуда он смотрит на мир в данный момент.

Когда гипотеза о создании системы спортивной аналитики только появилась, никто из нас, ее будущих разработчиков, не мог с уверенностью сказать, удастся ли воплотить ее с той степенью точности, которая соответствует серьезной аналитической работе. Прежде чем сортировать игроков по уровню мастерства, следовало собрать исчерпывающую статистику по каждому удару: траектории движения шаров, приложенной силе, сложности удара, углу резки, итоговому результату и множеству других параметров, влияющих на исход. Изначально задача формулировалась так, чтобы система работала на основе уже доступных в интернете записей и трансляций матчей, где видео варьировались по

качеству до полной непредсказуемости: от ракурса съемки и высоты камеры до стороны обзора, освещения и даже окружающего фона, который мог включать шумные трибуны или случайные помехи.

Собирать статистику требовалось независимо от этих факторов, создавая универсальный алгоритм, способный извлекать данные из хаотичного потока визуальной информации, словно очищая руду от примесей. Это подразумевало не только разработку надежных моделей распознавания, но и интеграцию методов компенсации искажений — от коррекции перспективы до фильтрации артефактов освещения, — чтобы обеспечить точность на уровне профессионального анализа. По мере тестирования на реальных партиях стало ясно, что такая система не просто автоматизирует рутинный труд судей и аналитиков, но и открывает дверь к новым горизонтам: от персонализированного обучения игроков до прогнозирования исходов матчей с вероятностью, граничащей с предвидением, и автоматического протоколирования (скоринга) бильярдных матчей в реальном времени. В итоге, преодолев начальные барьеры, мы получили инструмент, который не только фиксирует прошлое, но и моделирует будущее, делая его еще более точным отражением человеческого потенциала.

В результате применения на практике собственного аналитического инструмента, за полгода подготовки, я лично достиг звания кандидата в мастера спорта. Я записывал на видео все свои тренировки и загружал в систему. Она накапливала статистику и показывала все промахи. Модуль рекомендаций давал подсказки, на что сделать акцент в следующих тренировках. Практическая работа за бильярдным столом сопровождалась спокойным теоретическим разбором результатов по записям. Это дало свои плоды. В паре с Юрием Поповым 19 ноября 2023 года мы одержали победу на командном региональном первенстве по пулу 9ка. Мы с ним играем начиная с 2003 года, как в команде, так и против друг друга. Этот турнир принес нам обоим звание кандидатов в мастера спорта (мне повторно).

За время работы над проектом мы сформировали вокруг него международное сообщество энтузиастов из нескольких десятков тысяч человек. В 2021 году я получил свидетельство от Роспатента на базу данных «Статистика ключевых параметров бильярдных ударов», в которую вошли сотни тысяч аннотированных ударов. В том же году зарегистрирован товарный знак «PerformStars».

В 2023 году мой проект получил поддержку и статус резидента «Сколково». В том же году, по итогам 27-го Московского международного Салона изобретений и инновационных технологий «Архимед», наш проект победил в номинации «Лучшая инновационная образовательная тех-

нология: исследование, пособие, стартап»! Также в 2023 году, при неизменной поддержке моего делового партнера Владимира Марценюка, которому я обязан всеми своими достижениями, была учреждена компания ООО «Системы машинного зрения», где я занял должность заместителя генерального директора.

По результатам своей работы, в 2024 году я получил орден «За заслуги в изобретательстве» от «Всероссийского общества Изобретателей и Рационализаторов» (ВОИР).

В 2025 году я получил российский патент на изобретение: «Способ определения и визуализации траекторий бильярдных шаров на бильярдном столе»! Заявки на международный патент отправлены в США, Китае и Индонезии. В том же году мы разработали мобильное приложение для пользователей, что формально засвидетельствовано также патентом на промышленный образец его интерфейса, а публике оно было впервые показано на 19й Всемирной Бильярдной Выставке (GBE 2025) 10 – 12 мая, 2025 в выставочном комплексе China Import and Export Fair Complex, Гуанчжоу, Китай.

Если подвести итог, то за шесть лет практической работы машинное зрение стало для меня повседневной практикой, предметом множества исследований и открытий, способом творческой самореализации и делом всей моей жизни. Не только в теории, а на собственной практике я понимаю о чем пойдет речь ниже во множестве подробностей.

ЧАСТЬ 1

Хронология становления машинного зрения



1950-е: Ранние основы

С середины прошлого века изобретательская мысль передовых ученых и инженеров бросила вызов эпохе атомных и ракетных технологий, ставя перед поколением дерзкую и не менее амбициозную цель: вывести технологии автоматизации на новый уровень, а именно — создать синтетическую систему восприятия. Одним из первых, кто сумел сформулировать эту мечту в ясные исследовательские цели, был молодой англо-американский математик Оливер Селфридж — позже его назовут «отцом машинного зрения». В марте 1955 он выступает на конференции Western Joint Computer Conference с докладом «Распознавание образов и современные компьютеры», где, по сути, предлагает первую компьютерную систему оптического распознавания текста. Конкретная задача — научить машину выделять «значимые черты» буквы на фоне лишних

деталей, чтобы распознавать символ, даже если он написан криво или снят в плохих условиях. Селфридж подчеркивал цель, которая звучит актуально и сегодня: компьютер должен не просто сравнивать пиксели по заданным вручную локальным условиям, а автоматически улавливать инвариантную «самость» объекта — узнавать «А» как «А» в сотнях вариаций.

В контексте той эпохи эта цель выглядела абсолютно фантастической. Для понимания, типичный «рабочий» компьютер уровня IBM 704 оперировал всего лишь от 4 до 32 тыс 36-битных слов на магнитной памяти и выполнял порядка 12 тыс операций с плавающей запятой в секунду — ничтожные ресурсы по меркам машинного зрения. Первые алгоритмы должны были быть предельно экономными: извлечь немного хороших признаков из очень ограниченного потока данных и сделать это быстро. В этих условиях идея Селфриджа — опираться на устойчивые элементарные «черты» и собирать из них более сложные сигнатуры — была изначально научной гипотезой и инженерным вызовом, который казался практически невыполнимым.

В 1956 г. Массачусетский технологический институт (MIT) строит один из первых полностью транзисторных компьютеров TX-0 — по тем временам «гигантскую» машину с 64 тысячами 18-битных слов магнитной памяти; она занимала целое помещение и впервые позволяла работать интерактивно, с дисплеем на основе электронно-лучевой трубки, рисующим точки и линии. Она буквально **рисовала изображение точка за точкой**, заставляя световое пятно пробегать по координатам, которые вычислял компьютер. Это был прорыв для своего времени. Впервые человек мог **не просто печатать команды и получать цифры на бумаге, а видеть результат прямо на экране**. Причем TX-0 позволял рисовать простые линии и фигуры, а также интерактивно работать с ними с помощью переключателей и светового пера. Именно с TX-0 началась эпоха интерактивной графики и визуального программирования.

Интересный факт: когда в 1958 году студент MIT **Иван Сазерленд** впервые увидел TX-0 в работе, именно этот «осциллографический экран» вдохновил его спустя несколько лет на создание знаменитой системы **Sketchpad (1963)** — первого графического редактора с прямым управлением, где можно было рисовать и перемещать объекты на экране. Можно сказать, что от этого мерцающего пятна на ЭЛТ родилась вся современная компьютерная графика — от CAD до Photoshop.

Значимой вехой в истории машинного зрения стало появление первого цифрового изображения. Это произошло через два года после доклада, в 1957, в Национальном бюро стандартов США инженер Рассел Кирш получил первую в истории цифровую фотографию — снимок

своего младенца, оцифрованный в крошечную сетку 176 на 176 пикселей. Этот новорожденный младенец стал символом рождения всей технологии: теперь «глазам» машины появилось на что смотреть. Снимок Кирша быстро разлетелся по техническим лабораториям: кто-то учился на нем подавлять шум, кто-то — выделять на нем границы, а кто-то — как Селфридж — искать в этой зернистой сетке повторяющиеся структурные мотивы, пригодные для узнавания.



Первое цифровое фото

Селфридж вел активную просветительскую работу с широкой научной публикой. В 1959 он публикует работу, которая стала легендой — «Пандемониум: парадигма обучения». В ней Селфридж образно описал свою модель распознавания: множество простых «агентов» (он называл их «демонами») параллельно сканируют изображение, реагируя каждый на свою черту — ориентацию штриха, пересечение линий, кривизну. Чем лучше черта подходит, тем громче «кричит» соответствующий агент; «крики» суммируются на более высоких уровнях, где уже «демоны» более сложных образов решают, что именно перед нами — скажем, буква, цифра или их комбинация. Ироничное название «pandemonium» — в буквальном смысле «множество демонов» — скрывало очень современную идею конкурентного параллелизма и многоуровневой обработки признаков. Для эпохи вакуумных ламп и ранних транзисторов это была понятная и доступная архитектура: много простых тестов было легче организовать, это уменьшало тяжесть вычислений и давало на выходе большой каскад простых сигналов. Именно по такому принципу работают современные искусственные нейронные

сети, где каждый «демон» — это нейрон, громкость «крика» которого, определяется его весом.

Летом 1960 вместе с психологом Ульриком Найссером Селфридж написал для журнала *Scientific American* популярную статью «Распознавание образов машиной». Там он доступным языком объяснял, как машина может научиться видеть буквы и простые объекты, почему ей нужна «иерархия признаков» и каковы границы имеющихся тогда возможностей.

«Машина должна уметь распознавать части и отношения между частями. Узнавание целого образа невозможно без распознавания его компонентов.»

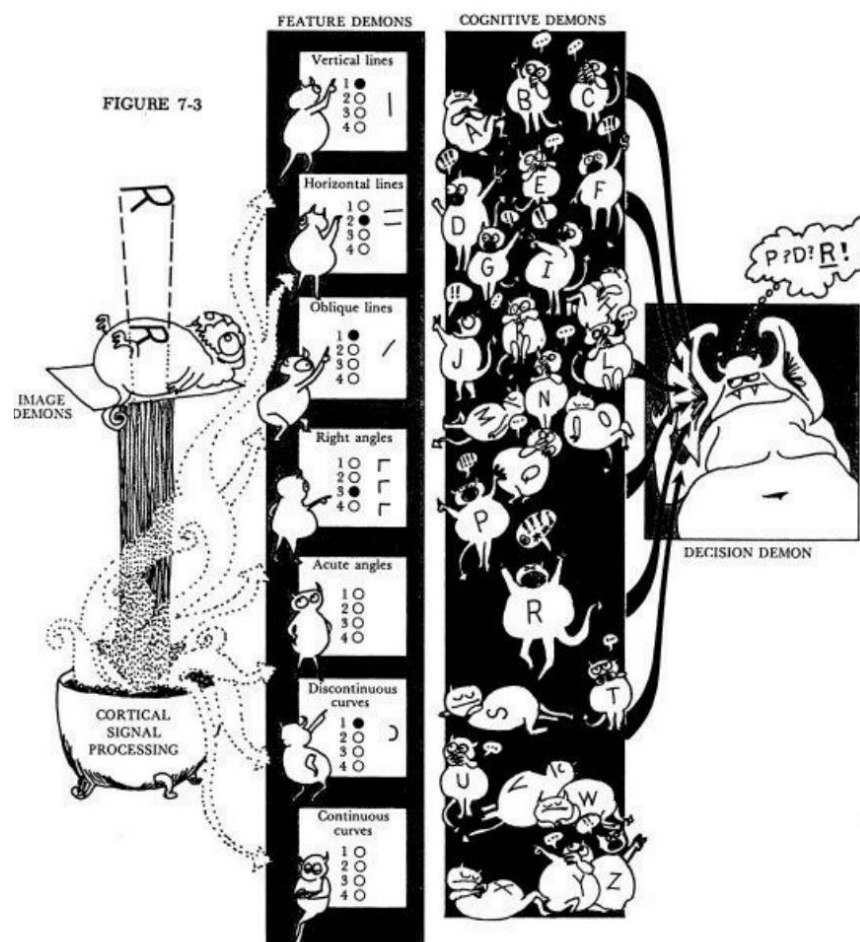
Эта мысль стала краеугольной для всей последующей архитектуры машинного зрения. Фактически, Селфридж здесь впервые сформулировал идею иерархии признаков — то, что через двадцать лет воплотилось в неокогнитрон Фукусимы, а затем — в сверточные сети Лекуна и Крижевского.

«Мы можем сделать машину, которая видит прямые линии, кривые или углы. Но пока никто не знает, как сделать машину, которая увидит кошку.»

Эта фраза — отражение всего состояния области на рубеже 1960-х. Ученые научились строить «детекторы» примитивных форм, но не могли соединить их в осмысленное восприятие. Именно поэтому Селфридж настаивал: машина должна «учиться» видеть мир многоуровнево, переходя от простых сигналов к сложным структурам — от линии к букве, от буквы к слову, от слова к смыслу.

Для читателей своего времени исследователи приводили и практические примеры вроде MAUDE — автоматического промышленного декодера азбуки Морзе, где программа последовательно проверяла входной сигнал по набору жестких условий. Это хороший и наглядный пример: когда есть узкая ниша, возможно установить все признаки вручную, чтобы машина прекрасно выполняла поставленную задачу; но чтобы выйти за ее пределы, ей нужно научиться конструировать иерархии признаков самой.

Самым ярким и запоминающимся примером в статье **Оливера Селфриджа и Ульрика Найссера «Pattern Recognition by Machine» (Scientific American, июль 1960)** было **распознавание напечатанных букв** — задача, которая тогда казалась почти чудом.



Авторы описывали, как машина может «научиться» видеть буквы не как цельные картинки, а как **набор признаков** — коротких линий, пересечений и дуг. Они объясняли, что распознавание буквы «А» возможно, если программа сначала находит две сходящиеся линии и перекладину, а затем «решает», что комбинация этих трех элементов соответствует символу. Буква «В», в их описании, состоит из вертикального штриха и двух дуг, одна над другой. Машина не хранит образ буквы целиком, она узнает его **по структуре частей и связей между ними**.

Этот пример в статье сопровождался иллюстрацией: на рисунке были показаны несколько простых символов, разложенных на примитивные элементы — линии и углы. Подпись объясняла, что «распознавание образа — это не запоминание картинки, а анализ ее структуры».

Селфридж и Найссер выбрали буквы именно потому, что они просты, но структурированы — идеальный полигон для демонстрации идеи **иерархии признаков**. Они писали: «Букву можно распознать по наличию и расположению нескольких линий и кривых. Машине не нужно хранить весь образ; она может принять решение на основании частей.»

Для своего времени это было революционно. Компьютер, который не просто «считал», а **разлагал картинку на смысловые элементы**, являлся первым шагом на пути к искусственному восприятию.

Любопытно, что в конце статьи они писали с характерным оптимизмом для того времени: когда-нибудь такие методы позволят машинам «видеть лица, узнавать рукописный текст и, возможно, понимать сцену» — все то, что начнет становиться реальностью всего лишь через тридцать лет, уже в начале 90х.

Как действующий специалист непосредственно по данной теме, я хорошо понимаю внутренние мотивы, которые двигали основоположниками технологии машинного зрения 60 лет назад. Эти мотивы живы в нас и по сей день — это борьба с эвристикой. До какой степени мы занимаемся оптимизацией и подгонкой наших алгоритмов под конкретные условия. И до какой степени мы можем обобщать, абстрагировать набор данных, которые мы даем машине для самостоятельного анализа. По сути ответ на оба вопроса — баланс времени и вычислительной мощности. Как вы сможете увидеть в этом обзоре, у каждого нового поколения исследователей стоит дилемма повышения производительности применяемых методов — использование локальной оптимизации и конкретизации алгоритмов, что быстрее и проще, но менее эффективно. Или, в перспективе ожидания роста вычислительной мощности, ослабление этой оптимизации в пользу универсальных методов автоматизации. Вероятностный подход дает больше информации в задачах, где невозможно добиться абсолютной точности.

1960-е: Формирование задач и первых алгоритмов

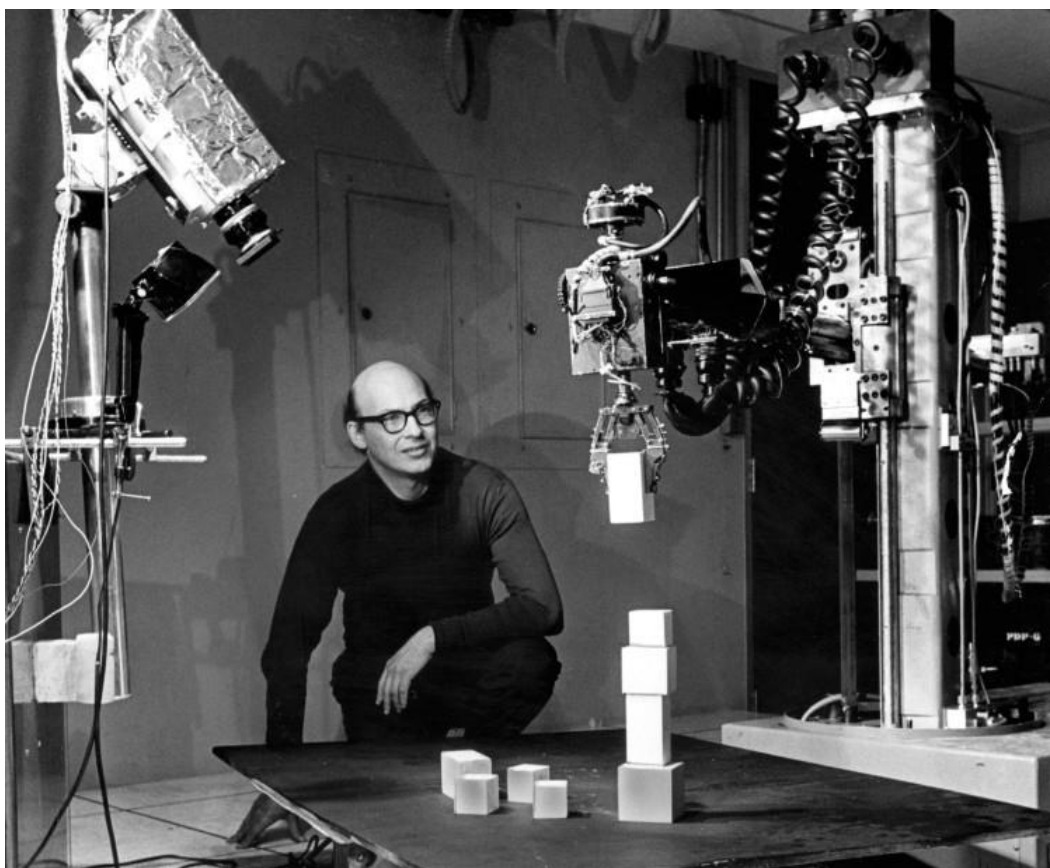
Уже в начале 1960-х в Москве рождается несколько фундаментальных идей, которые через десятилетия станут «скелетом» современного машинного обучения, в том числе и машинного зрения. В 1961–1963 Михаил Цетлин формулирует концепцию «обучающихся автоматов» — простейшие стохастические устройства, накапливающие опыт по схеме «награда/штраф». Это общий для ИИ механизм, и его теоретическое описание впервые задает направление для дальнейших практических изысканий: можно учить систему действовать в среде без полной модели мира.

В 1963 Владимир Вапник и Алексей Червоненкис показывают алгоритм «обобщенных портретов»: способ классификации точек оптимальной гиперплоскостью. Для инженера прикладника 60-х это способ более универсальной классификации образов в шуме. По сути это ранний прототип метода опорных векторов (Support Vector Machine — SVM). Позже та же школа сформулирует теорию Вапника-Червоненкиса (VC-теорию) и принцип минимизации структурного риска, которые стали стандартом качества для статистического обучения и напрямую повлияли на алгоритмы машинного зрения 1990-х.

Параллельно, в 1964, М. А. Айзерман, Э. М. Браверман и Л. И. Розоноэр публикуют «Метод потенциальных функций»: переводят обучающуюся задачу в пространство признаков, где линейная разделимость достигается «неявно» — то, что позже мир назовет «kernel trick» (трюк с ядрами). Это был способ решать сложные задачи распознавания под жесткие аппаратные ограничения, и это еще один прямой предок SVM/ядерных методов, широко использованных в классическом машинном зрении до эры глубокого обучения.

Киевская школа Алексея Григорьевича Ивахненко в 1965–1968 годах запускает GMDH — «групповой метод обработки данных»: многошаговое самосогласованное построение полиномиальных моделей, которое мы сегодня без колебаний называем одним из первых работающих алгоритмов глубокого обучения. В прикладном машинном зрении тогда это использовали для задач идентификации и прогноза, где ручных признаков мало, а шум велик; метод задает цель автоматического выбора структуры модели под данные.

На другом конце земного шара, в Массачусетском технологическом институте, в 1963 Лоуренс Робертс защищает диссертацию «Machine Perception of Three-Dimensional Solids». Идея заключалась в том, чтобы превратить обычную фотографию в аккуратный линейный рисунок — набор ребер и контуров, а затем восстановить из этих линий объемные полигоны, «блоки» сцены и понять их взаимное расположение. В терминах той эпохи это означало научить компьютер выполнять то, что художник-чертежник делает рукой: увидеть твердое тело сквозь его проекции и скрытые линии. Робертс не просто наметил концепцию, он описал и запрограммировал целую цепочку процедур (пайплайн) — от выделения линий на снимке до реконструкции 3D-структуры. Для машин того времени это был прорыв: появилась программа, которая, глядя на плоскую фотографию, возвращала не набор пикселей, а схему мира из объемных тел. Так родился «блоковый» взгляд на сцену — привычный нам сегодня язык 3D-графики и инженерного зрения.



(Марвин Минский в лаборатории МТИ, 1968 г.)

Через три года, летом 1966, в соседней лаборатории Марвин Минский дает студентам дерзкое для своего время техзадание: «подключить камеру к компьютеру и заставить систему описывать то, что она видит». Этот «летний проект по машинному зрению» вошел в историю не столько результатами, сколько масштабом амбиций. В меморандуме июля 1966 года Сеймур Пейперт прямо формулирует цели: научиться делить картинку с видисектора (телевизионной трубки) на области - «фигура» и «фон», описывать регионы по форме и поверхностям и, в финале, идентифицировать объекты, сопоставляя их со «словарем известных объектов». План был ступенчатым: как можно быстрее собрать работающий набросок системы и по нарастающей усложнять ее — этот оптимизм отлично передает дух времени. Летний проект середины шестидесятых затянулся на десятилетия.

Сеймур Пейперт — южноафриканский математик и один из пионеров ИИ, сооснователь (вместе с Марвином Минским) легендарной лаборатории искусственного интеллекта МТИ. Он прославился и фундаментальной критикой перцептронов (книга *Perceptrons* в соавторстве с Минским), и работами об обучении (язык Logo, конструкционизм). В 1966 Пейперт курировал «летний проект» и написал тот самый меморандум, а Минский как руководитель лаборатории задавал общие контуры работы.

Коротко, суть фундаментальной критики перцептронов в том, что Минский и Пейперт показали пределы «классического» перцептрона как линейного порогового элемента.

Перцептрон в их книге — это функция, где есть одна линейная комбинация входов и порог. Для таких моделей они аккуратно доказали, что, если классы не разделимы одной гиперплоскостью, перцептрон этого не выучит в принципе. Поэтому задачи вроде «исключающего ИЛИ» (XOR), «четности» (parity — четное/нечетное число единиц) или некоторых простых топологических свойств изображений неразрешимы для однослойного перцептрона. Это был удар не по нейросетям вообще, а по существовавшим тогда завышенным надеждам на одиночный линейный узел.

Дальше они рассмотрели «усиления» перцептрона, которые предлагались для обхода барьера линейной разделимости. В этой работе они показали еще один важный предел: свойства, зависящие от глобальной структуры изображения, вроде связности фигуры, требуют либо нелокальных связей, либо экспоненциального роста «порядка» (комбинаторных взаимодействий входов). Иначе говоря, если вы жестко ограничиваете архитектуру локальностью и простыми суммами, то некоторые естественные зрительные задачи недостижимы при разумных ресурсах.

Наконец, они трезво отметили инженерный вакуум того времени: для многослойных сетей не существовало работоспособных алгоритмов обучения, а гарантия сходимости классического правила Розенблатта действовала только в линейно-разделимом случае. Поэтому «добавить слоев» в 1969 году означало скорее усложнить схему без способа ее обучить.

Исторический эффект книги — остывший оптимизм вокруг однослойных моделей и перенос внимания к более выразительным архитектурам и алгоритмам. Когда спустя десятилетия появились градиентные методы обучения многослойных сетей и большие корпуса данных, часть ограничений была обойдена: многослойные сети представляют XOR и «глобальные» свойства. Но главный вывод «Perceptrons» никуда не делся: у модели должна быть достаточная выразительность и правильные связи, а ее возможности строго зависят от архитектурных ограничений, которые в нее закладываются.

Реальность, разумеется, оказалась упрямее. Камеры давали нестабильный, шумный сигнал; компьютеры были медленными и крошечными по памяти; любой шаг вроде детектирования границ требовал аккуратного баланса между чувствительностью и устойчивостью. Но именно эти попытки — пусть и «слишком смелые на одно лето» — консолидировали вокруг проекта сообщество инженеров: нужно научиться

видеть объекты на фоне, удерживать геометрию сцены, работать со временем, когда речь о видео, и постепенно переходить от пикселей к смыслам.

Параллельно развивалась технология распознавания лиц. В середине 1960-х Вудроу (Вуди) Бледсо в Panoramic Research в Пало-Альто запускает серию первых экспериментов. Тогда еще не было удобных сканеров и меток на изображениях, и систему приходилось кормить числами вручную: оператор измерял на фотографии координаты ключевых точек — уголков глаз и рта, переносицы, контура подбородка — а программа сравнивала геометрию этих «масок» между снимками. В отчетах 1964–1966 годов Бледсо и коллеги документируют рост масштаба экспериментов и точности на все более разнообразных наборах портретов; а сам подход называли «человеко-машинным», подчеркивая разделение труда между человеком, который метит признаки, и машиной, которая принимает решение. Это был первый шаг к биометрии — способность программно «видеть» лицо, и узнавать человека.

В 1971 году Алан Дж. Голдстейн, Леон Д. Хармон и Артур Б. Леск из Bell Labs опубликовали в *Proceedings of the IEEE* статью «Identification of Human Faces»: они сформулировали архитектуру решения — представить лицо как вектор признаков и искать совпадение статистическими методами — и показали на реальных данных, что это работает. Их вектор включал ~21 субъективно оцениваемый признак (вроде толщины губ, оттенка волос, длины ушей), а сами оценки давали люди-наблюдатели; дальше машина ранжировала базу по «близости» описаний и выдавала кандидатов. Это по сути тот же «человеко-машинный» подход Бледсо, но оформленный как воспроизводимое исследование в ведущем журнале — с описанной процедурой, метриками и выводами.

И наконец — аппаратная развязка, без которой «глазам» машин просто не на что было бы смотреть. В 1969 году в Bell Labs физики Уиллард Бойл и Джордж Смит изобретают ПЗС-датчик — (сокр. от «прибор с зарядовой связью» CCD charge-coupled device). В учебном объяснении это была решетка маленьких чувствительных ячеек, каждая из которых накапливает заряд под действием света; потом эти заряды «перекладываются» по цепочке и считываются как цифровое изображение. Революционность ПЗС была в том, что он наконец переводил свет в цифру без промежуточной химии: появлялись действительно электронные глаза. За это открытие Бойл и Смит получили Нобелевскую премию по физике в 2009; в воспоминаниях коллег прозвучал даже анекдот, что ключевая концепция была сформулирована буквально за час — на фоне конкури-

рующего проекта, связанного с цифровой памятью. История изящно иллюстрирует дух Bell Labs: искали одно, нашли другое, но вовремя распознали, насколько «другое» огромно.

Уже в 1975 инженер Kodak Стив Сэссон собирает первый переносной цифровой фотоаппарат: черно-белая картинка 100×100 пикселей с экспериментального ПЗС писалась на кассету целых 23 секунды, а для просмотра требовался специальный проигрыватель, который «растягивал» изображение до телевизионной развертки. Камера весила как небольшой тостер и питалась от 16 никель-кадмиевых аккумуляторов — но это был действующий прототип принципиально позволявший фотографировать без пленки, сразу в цифру. Для компьютерного зрения это означало приток реальных изображений и видео, который уже нельзя было остановить.

1970-е: Новые методы анализа изображений

В начале 1970-х в лабораториях по всему миру начали учиться видеть на изображениях простые формы — линии, окружности, дуги. В 1972 году исследователи из Stanford Research Institute Ричард Дуда и Питер Харт предложили элегантный алгоритм, который сделал поиск геометрии устойчивым к шуму: «преобразование Хафа». История тут немного парадоксальная: сам метод был запатентован инженером Полом Хафом в 1962, но именно работа Дуды и Харта десять лет спустя превратила идею в практичный инструмент компьютерного зрения. Хаф первым предложил мыслить не самой линией на картинке, а ее **параметрами**. Дуда и Харт довели это до удобной формы: они описали лю-

бую прямую через **угол и расстояние от начала координат**. Такая запись одинаково хорошо работает и для горизонтальных, и для вертикальных, и для наклонных прямых.

Дальше — фокус. Представьте, что на изображении вы заранее нашли «подозрительные» точки контура. Каждая такая точка говорит: «через меня могут проходить вот такие линии» — и перечисляет все подходящие пары «угол-расстояние». Мы кладем большой «лист» с клеточками по этим двум параметрам и просто **считаем голоса**: каждая точка добавляет по единице всем клеткам, которые ей подходят. Там, где в итоге накапливается явный максимум голосов, — и есть параметры реальной линии в изображении. То есть мы как бы переносим задачу из «грязной» картинки в чистое параметрическое пространство, где нужная прямая проявляется как пик в простейшей таблице.

Почему это работает в шуме? Потому что шум «размазывает» свои голоса по разным клеткам, а настоящая линия заставляет **многие** точки проголосовать **за одно и то же** место. В результате пик все равно вылезает, даже если часть контура пропала или затерялась. Именно эта простая «статистическая» логика и сделала метод практичным: не надо пытаться вычерчивать идеальную линию прямо на изображении — ее легко найти как максимум голосов в таблице параметров.

Главное достоинство — шум и прорехи не мешают: даже если половина пикселей пропала, оставшиеся все равно «сойдутся» на правильной прямой. Позже тот же прием распространили на окружности и прочие кривые. Для инженеров 1970-х это позволило брать сырое изображение с камеры и автоматически вылавливать на нем полосы, разметки, кромки деталей, шестеренки и винты по характерным дугам. На реальных машинах, где память измерялась десятками килобайт, а кадр приходилось считывать из аналогового видеосигнала построчно, такая надежность значила очень много.

Параллельно формировалась более «когнитивная» линия — как представлять в компьютере сложный объект, не сводя его к одной линии или контуру. В 1973 году Мартин Фишлер и Роберт Элшлагер предложили идею «структурных образов»: рассматривать вещь как набор частей с допустимыми связями между ними. Лицо — это не один силуэт, а глаза, нос, рот и их относительные положения. Такая постановка вопроса позволяла искать на фотографии не эталонные шаблоны, а соотношения простых элементов, учитывая, что части могут сдвигаться, немного вращаться, а кое-где и прятаться за шумом или тенью. С практической стороны это был ответ на реальность: камеры дрожали, освещение менялось, люди поворачивали головы — и требовалась модель, которая терпимо относится к вариациям. Позже этот взгляд «объект = части + связи» прорастет в множество методов — от распознавания лиц

до поз человека, но в начале 1970-х это был естественный переход от жестких шаблонов к более гибкому описанию образов.

Семидесятые подарили еще одну важную идею, без которой многие прикладные системы просто не заработали бы: работа с текстурой. В том же 1973 Роберт Харалик с соавторами показал, как извлекать из серого «зерна» изображения числовые признаки — например, насколько часто соседние пиксели бывают похожими или, наоборот, резко контрастными. Эти текстурные алгоритмы нашли широкое применение во многих задачах, например в дистанционном зондировании Земли, или в промышленной инспекции. В эпоху, когда камеры давали далеко не идеальную картинку, а вычислители были медленными, только математика служила надежным компенсатором несовершенства сенсоров.

А еще именно в семидесятые появились цветные оптические матрицы. До этого цветная телевизионная камера сочетала громоздкую оптику и **приемные электронно-лучевые трубки**, по одной на каждый канал — красный, зеленый, синий. В 1974 году ученый Eastman Kodak Брайс Байер предложил разложить светофильтры по поверхности фотоприемной матрицы с перевесом в пользу зеленого: ровно половина ячеек — зеленые, оставшаяся половина делится пополам между красными и синими. Так рождается знакомая 2×2-плитка RGGB: сверху, скажем, красный и зеленый, снизу — зеленый и синий, и дальше эта плитка повторяется до краев ПЗС (CCD, *charge-coupled device*) или КМОП (CMOS, *complementary metal-oxide-semiconductor*). На ней каждый пиксель — это маленький фотодиод, который ловит фотоны, превращает их в электрический заряд, а потом электроника построчно считывает этот заряд и переводит в цифровые числа.

Зачем удваивать зеленый? Потому что наш глаз острее всего чувствует **яркостные** детали, а вклад в яркостный сигнал идет главным образом через зеленый канал. Увеличив число «зеленых проб» в два раза, Байер добился того, что камера фиксирует тонкую фактуру и контуры резче, даже если красного и синего замеров вдвое меньше. Это компромисс эпохи практиков: сохранить максимум видимой четкости при минимуме аппаратной сложности — один чип и дешево изготавливаемая цветная маска сверху.

Другой компромисс заключается в том, что **в каждой точке сенсора измерен лишь один цвет**. Двух остальных цветов нет — их приходится восстанавливать вычислением из соседей. Этот этап называют «дебайеризацией» или демозаикой. Если совсем по-честному, это умная раскраска по контуру: алгоритм смотрит вокруг пикселя, пытается понять, есть ли рядом граница, куда тянется узор, где текстура, и по этим подсказкам аккуратно «дорисовывает» недостающие каналы. На гладких участках хватает простой интерполяции; возле контрастных кромок

приходят на помощь более сложные методы, которые «держат» контуры, не размывая их цветными ореолами. Здесь и прячутся типичные артефакты ранних цифровых камер. Иногда вдоль тонких линий появляется «молния» — ступеньки и разноцветные каймы. На мелких повторяющихся деталях (ткань, сетка) — цветное муаре. Против этого долго ставили оптические низкочастотные фильтры — тонкие пластинки перед сенсором, которые едва заметно «размазывают» картинку, чтобы не возникал ложный цвет. Позже, когда алгоритмы стали умнее и пиксели — мельче, многие производители от фильтра отказались ради лишней резкости, переложив борьбу с муаре на программы.

Еще одна деталь, которую обычно не проговаривают. Цветные фильтры «съедают» свет: красный под красным, синий под синим — всегда темнее зеленого, да и сами красный/синий каналы у сенсора шумнее. Поэтому камера применяет **баланс белого** — по сути, поднимает одни каналы и заглушает другие, чтобы белое в кадре стало по-настоящему белым. Подъем канала поднимает и его шум — отсюда знакомая всем «грязь» на синем небе в сумерках. Дальше идет цветокоррекция: матрицей преобразования сырые данные сводят к стандартному цветовому пространству — чтобы трава была травой, а кожа — кожей на разных устройствах.

После изобретения цветного оптического сенсора, в мире машинного зрения произошел качественный сдвиг: алгоритмы стали получать не только формы и границы, но и цветовые измерения, которые можно учитывать при сегментации и распознавании.

1980-е: Алгоритмы и нейросети – от теории к практике

В восьмидесятые машинное зрение делает важный поворот от разрозненных трюков с изображениями к целостным моделям восприя-

тия и устойчивым инженерным методам. В Японии десятилетие начинается с дерзкой конструкции Кунихико Фукусимы. Его «неокогнитрон» вырастает из нейрофизиологии — из наблюдений о том, как нейроны зрительной коры реагируют на простые штрихи и как более сложные реакции возникают из сочетаний простых. Фукусима показывает, что можно построить многоуровневую сеть, которая сначала улавливает элементарные детали, затем «собирает» их в более сложные мотивы, а на верхних уровнях распознает целый образ и при этом не путается от небольших сдвигов или растяжений рисунка. По сути это раннее, почти предсказательное воплощение того, что спустя годы станет стандартом для распознавания изображений. Важная деталь эпохи: автор сознательно ставит задачу «как у человека» — не только получить верный ответ, но добиться устойчивости к неизбежной вариативности реального мира. Статья Фукусимы выходит в 1980 году в журнале *Biological Cybernetics* и мгновенно дает исследователям новый язык описания зрительных задач — язык иерархий и инвариантностей.

Преобразование Хафа дало возможность оцифровывать геометрию на изображениях. В восьмидесятых были достигнуты естественные пределы этого метода: он хорош, когда модель проста (линия, окружность), параметров мало, а «правильных» точек заведомо больше, чем мусора. В реальных сценах это условие часто рушилось: кадр дрожит, разметка неточна, блики и шум дают горы ложных срабатываний, да еще и сама модель усложняется — вместо линии приходится подгонять плоскость, вычислять перспективное соответствие между двумя изображениями или другую многопараметрическую зависимость. Здесь голосование на фиксированной решетке параметров уже не вытягивает: решетка либо груба и промахивается, либо тонка и «тонет» в шуме, причем одновременно.

RANSAC

В 1981 Мартин Фишлер и Роберт Боллс из SRI предлагают метод, который сегодня есть в каждом учебнике: *Random Sample Consensus*, «случайный выбор с согласованием» RANSAC — как бы «стохастический кузен» метода Хафа, но без его сетки и с гораздо более широкой областью применения. Идея проста и красива — взамен аккуратного перебора всех возможных параметров он предлагает играть в быструю лотерею: много раз наугад брать минимальный набор опорных точек, по нему грубо восстанавливать модель (будь то прямая, плоскость или проективное преобразование между кадрами) и затем спрашивать у всей выборки: «кто с этой моделью согласен в пределах разумной погрешности?». Если таких «согласных» набирается много, значит, мы «попали» в правду; если нет — снова «бросаем кости». Смысл в том, что выбросы

не соберут вокруг себя большой консенсус, а вот настоящая геометрическая зависимость — соберет, даже если «правильных» точек меньшинство и они разбросаны по изображению. Эта процедура оказалась мощнейшим стабилизатором зрительных алгоритмов: стало возможно работать с реальными изображениями, где есть блики, шум, нестыковки разметки, а камера дрожит. Метод появился при поддержке DARPA (англ. Defense Advanced Research Projects Agency) — Управление перспективных исследовательских проектов Министерства обороны США: Пентагон вкладывал деньги в то, чтобы автоматический анализ сцен выдерживал шумы, неизбежные вне лаборатории.

Оптический поток

В том же 1981 году появляется еще один «кирпич» машинного зрения, но уже в задачах анализа видео — метод оптического потока. Брюс Лукас и Такео Канаде описывают метод, который в дальнейшем позволит системам замечать движения в кадре не по «разности двух картинок», а аккуратно — через сопоставление маленьких окон изображения. Принцип похож на ручную подгонку: выбираете паттерн на первом кадре и находите, где он оказался на втором, минимизируя разницу яркостей. Повторяете для многих окон — и получаете поле смещений, то есть оценку оптического потока: куда и как быстро все движется. Сильная сторона метода — в его локальности и итеративности: он достаточно легкий вычислительно, чтобы его можно было реализовывать на машинах того времени, и достаточно точен, чтобы им можно было управлять роботами, стабилизировать видео или отслеживать объекты. У машинного зрения появляется новый класс задач - трекинг видео. Алгоритмы начинают анализировать изменения в потоке изображений, а не только геометрию на кадре.

В 1982 году Дэвид Марр в своей знаменитой книге «*Vision*» предлагает теоретический подход: любые машинные алгоритмы надо понимать на уровне целей («что мы хотим извлечь из изображения?»), представлений («в каком виде это хранить и преобразовывать?») и механизмов («какими вычислениями это достигается?»). До него многие публиковали удачные приемы: показали, «красиво стало» — и достаточно. После «*Vision*» рецензенты, коллеги, собственные студенты стали спраши-

вать: а какова цель вычисления, в каком представлении вы держите результат, чем ваши механизмы обоснованы физикой и геометрией? Он вывел дискуссию из чисто практической плоскости «вот фильтр — вот картинка» в плоскость теоретического целеполагания и уровней описания. Этого «холодного душа» особенно ждали именно инженеры-практики: им нужен был новый семантический аппарат.

Говорить, что Марр «ввел современную классификацию задач» в духе сегодняшних названий — детекция объектов, классификация, сегментация, трекинг, поза, текст — было бы неточно. Он не рисовал такую таблицу. Его вклад в другом: он собрал теоретический каркас. Поэтому, когда позже появились названия и бенчмарки, они органично легли на его семантический аппарат.

Марр вводит три **уровня объяснения**: вычислительный (что именно извлекаем из изображения и по каким принципам это вообще возможно), алгоритмико-репрезентационный (в каких представлениях и какими процедурами это делаем) и реализационный (как это воплощается физически — в нейронной ткани или в железе). Далее — **три стадии внутреннего зрительного представления**, через которые, по Марру, проходит сигнал. Сначала «первичный набросок»: локальные края с точной позицией и ориентацией, без мусора и двойных срабатываний. Это не «готовая сегментация», а инвентаризация элементарных событий изображения: края и их ориентации, окончания штрихов, Т- и L-сочленения, «пятна» (blobs), — все это получено корректной для изображения физикой — фильтрацией и операторами, а не произвольными трюками. В работах Марра–Хилдрейт это аккуратно связывается с гауссовым сглаживанием и нулевыми пересечениями лапласиана: откуда берется край и почему он должен обнаруживаться так, а не иначе.

Следующим выполняется **два с половиной мерный набросок** — представление поверхности сцены в координатах наблюдателя: локальная глубина (или относительная дистанция), ориентация участков поверхности (нормали), разрывы по глубине и по отражающим свойствам. Здесь могут встречаться любые «подсказки» — стерео, движение, тени, текстура, — но результат все еще «привязан к взгляду»: это не имя объекта, а осмысленная карта видимой поверхности.

И только затем — **трехмерное представление**: уже не «то, как нам это видится», а **объектно-центрическое** описание, независимое от ракурса. В семантике Марра–Нисихары это иерархия объемных примитивов (вроде обобщенных цилиндров) и их отношений, из которых можно восстановить один и тот же предмет при изменении точки зрения.

Если аккуратно привязать к этому современный словарь, то, что мы сегодня зовем выделением границ, углов, локальных «пятен» и вообще ранним разделением «фигура–фон», — относится к **первичному**

наброску. Оценка глубины, нормалей, оптического потока (от стерео, движения, тени, текстуры) — это содержимое **двух с половиной мерного наброска**, потому что речь о видимой поверхности в системе координат наблюдателя. Узнавание объекта, его частей и иерархий, поз и взаимодействий — уже территория **трехмерного представления**, где представление должно переживать смену ракурса. Семантическая и инстанс-сегментация в сегодняшнем смысле стоят на границе: их «маски» по форме ближе ко второму уровню (поверхности), но их смысловые метки и «имена» — шаг к третьему.

К примеру, в моем бильярдном проекте эта иерархия Марра оживает как последовательная цепочка алгоритмов, где каждый этап последовательно обрабатывает входящий сигнал и строит все более абстрактное и инвариантное представление. Мы продвигаемся от сырых пикселей к семантике удара, демонстрируя, как вычислительная теория машинного зрения может стать практическим инструментом в условиях реальных трансляций с их хаотичными ракурсами, разнообразным оборудованием и настройками освещения. На вычислительном уровне наша цель — извлечь траектории шаров и их взаимодействия в нормализованной системе координат стола, где законы физики (геометрия проекции и динамика движения) ограничивают пределы их возможного поведения. Алгоритмически это разворачивается через первичное представление, где нейросетевая семантическая сегментация объектов на кадре распознает края игровой поверхности среди множества сопутствующих элементов - углов бортов, игроков, пятен бликов или теней.

Переходя к двух с половиной мерному представлению, YOLO-детектор объектов локализует шары на видимой поверхности, оценивая их относительные позиции и «глубину» в плоскости кадра через рамки, опираясь на подсказки вроде текстуры сукна или теней от кия, — это еще не полная сцена, а карта игровой поверхности, привязанная к взгляду камеры. Наконец, на этапе трехмерного представления происходит трансформация: зная координаты четырех углов стола из первичного представления, я применяю гомографию для проекции координат шаров из пиксельного пространства кадра в нормализованную 2D-систему стола (ширина и длина у бильярдных столов стабильны), добавляя третье измерение — время удара по номеру кадра, — что рождает объектно-центричное описание: иерархия примитивов (шары как точки) с отношениями (траектории, столкновения), независимая от ракурса и готовая к семантике — распознаванию удара как последовательности векторов скоростей. Реализационно это воплощается в Caffe или PyTorch на GPU.

Вернемся к концу 1970-х — началу 1980-х, когда ядро машинного зрения было совсем компактным: выделение краев и примитивов, сегментация на области, «выделение формы» (оттенки, текстура, контуры),

простые трехмерные модели объектов и первые постановки конкретных задач распознавания. Это была горстка фундаментальных направлений — шесть–восемь, внутри которых шла детальная работа. Уже в 1990-х появится устойчивое «распознавание лиц» и документный анализ, а к середине 2000-х появятся признаки локальных точек, сцепление нескольких видов информации, первые зрелые трекары. Сегодня существуют десятки четко выделенных задач, и каждая имеет свои наборы данных и метрики. Детекция может использоваться отдельно от классификации, сегментация делится на семантическую, инстанс- и паноптическую, трекинг — на одиночный и многообъектный, поза — на 2D и 3D, глубина — на стерео, монокулярную и мультивидовую; к этому добавились видеопонимание и распознавание действий, сценографы (scene graph) и «понимание отношений», выделение и чтение текста в естественных сценах, реидентификация людей и объектов, построение 3D-сцен, оптический и «сценовый» поток, генерация и редактирование изображений, мультимодальные задачи вроде подписей к картинкам и «вопрос–ответ по изображению». Появилась целая «методическая надстройка»: самообучение и дистилляция признаков, надежность и безопасность, перенос между доменами, экономия вычислений на конечном устройстве.

Любой современный пайплайн — от детектора в телефоне до трансформера, который одновременно читает и «смотрит», — все равно пишется по марровской схеме: сначала формулируется цель (что именно извлекаем и какие инвариантности нам нужны), затем выбирается представление (карта признаков, маска, глубина, токены-патчи), и только после этого обсуждаются механизмы (архитектура, потоки данных, ускоритель). В этом смысле Марр «заставил» всю дисциплину говорить на одном языке. И именно поэтому сегодня задач больше на порядок, а спорим мы все о том же: что считать правильным представлением и где проходит граница того, что из изображения вообще можно извлечь. Сам Марр умер за два года до своей публикации; монографию коллеги довели до печати уже посмертно.

Детектор контуров Кэнни

На этом фоне Джон Кэнни в 1986 делает выдающийся алгоритм: выделяет критерии границ и контуров на изображении и получает из них

детектор, который на десятилетия станет базовым инструментом машинного зрения. На уровне вычислительной постановки по-марровски цель проста: из непрерывного фотосигнала извлечь места, где с высокой вероятностью меняется **поверхностная структура** — там, где на изображении возникает скачок яркости, за которым обычно стоят границы объектов или их частей. Представление, к которому мы хотим прийти, — это **элементы «первичного наброска»**. Механизм — тот самый «оптимальный» анализ Кэнни: сначала физически корректное сглаживание, чтобы не плодить ложные детали; затем оценка направления наибольшего изменения яркости; затем «подавление немаксимумов» вдоль этого направления, чтобы край стал тонким и одинарным; и, наконец, двойная пороговая схема, которая протягивает слабые участки там, где они **структурно связаны** с сильными. Поэтому на реальных, шумных кадрах алгоритм Кэнни является очень эффективным, предоставляя связный набор контурных элементов, пригодный для группировки, измерений и последующего анализа.

К концу восьмидесятых от сырых контуров переходят к **организации границ в осмысленные линии**. Активные контуры Касса–Уиткина–Терзопулоса (1988) — это пример контурного анализа. На вычислительном уровне цель звучит так: найти **гладкий, связный контур**, который проходит через те места на изображении, где есть достоверные признаки границы, и при этом не «ломается» на шуме. Представление — не пиксельная маска, а **параметрическая кривая** (упругая «веревка»), у которой есть внутренняя энергия гладкости и внешние силы изображения, тянущие ее к реальным краям. Механизм — итерационная эволюция этой кривой до минимума энергии: положили примерно на объект, отпустили — и кривая подтянулась к истинному контуру. Важно проговорить точность терминов: «змейка» — это **границе-ориентированная** (boundary-based) процедура; она дает аккуратные линии, превращая их в **целостные границы поверхности** — тот материал, из которого последующая аналитика строит **промежуточное** представление: разрывы по глубине, очертания органов на медицинском снимке, силуэт детали в САПР системе. В классическом виде «змейка» сама по себе **не стартуует «из ниоткуда»**. Ей нужна начальная кривая — грубый контур, лежащий *рядом* с истинной границей, плюс разумный масштаб и параметры гладкости. Дальше уже включается автоматика: кривая итеративно «едет» по полю изображений к ближайшей устойчивой границе. Но если изначально положить ее далеко, она прилипнет к первой сильной кромке — и это может быть не тот объект. Поэтому в конце 80-х «змейки» задумывались как **полуавтоматическая** сегментация: человек задает *инициализацию* (кольцо вокруг органа на снимке компьютерной томографии, петля вокруг детали на фото), машина доводит до физически правдоподобной формы. Для вычислительных мощностей того времени это был

оптимальный компромисс: немного работы оператора сэкономило много вычислений и спасало от локальных минимумов.

Позже появились модификации, уменьшающие зависимость от ручной работы: «надувная» сила, которая раздувает контур изнутри; геодезические и «градиентно-векторные» версии, лучше подтягивающиеся к слабым и разрывным краям; регион-бейзед-модели (типа Chan–Vese), которые ориентируются не только на резкие кромки, но и на однородность внутри/снаружи. Но и они почти всегда требуют **зерна** — точки, отрезка, грубого прямоугольника. Иначе алгоритм не знает, *какую именно* из множества границ сцены вы имели в виду.

В современном конвейере это выглядит так: старт может дать детектор (рамка объекта), карта краев или даже ручная отметка; дальше активный контур автоматом доводит границу, делая ее гладкой и топологически аккуратной. А в задачах, где есть интерактивность (медицина, дизайн формы, правка масок), ручная работа остается разумной частью метода: вы показываете системе *куда смотреть*, она делает остальную работу.

Ограничения ранней эпохи машинного зрения — от медленных процессоров и ограниченной памяти до шумных видеосигналов и дефицита масштабных данных — не становились непреодолимыми барьерами, а напротив, формировали дисциплину разработки, побуждая к созданию компактных, интуитивно понятных и устойчивых алгоритмов. Парадоксальным образом именно эти рамки обеспечили их долгожительство: многие из них продолжают служить основой современных систем, интегрируясь как надежные модули в сложные архитектуры, где они демонстрируют свою проверенную ценность в условиях реальных задач.

1990-е: Статистические методы, функции и данные

В 1991 стартует русскоязычный (с англоязычной версией) журнал *Pattern Recognition and Image Analysis* (PRIA). Он стал главной площадкой для школы распознавания образов в России/СНГ, где публиковались и математические основы, и прикладные работы по обработке изображений. Журнал выходит и сейчас; бумажные выпуски ведутся с 1991 года, а англоязычная версия индексируется у Springer. Это важный медиа ресурс для статей, рецензирования и обсуждения стандартов качества.

В том же 1991 запускается ежегодная конференция **GraphiCon** — крупнейший форум по компьютерной графике и машинному зрению в регионе. Конференция пережила тяжелые девяностые без пропусков и стала местом, где регулярно встречались академики, вузы и индустрия.

Eigenfaces

«Метод собственных лиц» (англ. **Eigenfaces**) стал известен в 1991 году благодаря работе Мэттью Терка и Алекса Пентленда в MIT: они сформулировали цель — **распознавать лица почти в реальном времени** — и довели идею до полноценной реализации. В более общем плане это распознавание на основе **РСА** (Principal Component Analysis, анализ главных компонент); тот же подход часто описывают как **преобразование Карунена–Лоэва (KLT)**, а само представление называют «пространством лиц» (*face space*).

Предпосылки для этого проекта появились чуть раньше: **Лоуренс Сирович** и **Майкл Кирби** в 1987 году показали, что лица удобно описывать в **низкоразмерном линейном подпространстве**, заданном главными компонентами большого набора выровненных портретов. Эти главные направления вариаций и получили имя «**собственные лица**»: полупрозрачные «маски», из которых любое конкретное лицо получается как их взвешенная сумма. Сравнить людей — значит сравнивать **координаты** (веса) этой проекции.

Главным достоинством такого подхода стала вычислительная оптимизация. Прямо раскладывать ковариацию «по пикселям» невозможно: у снимка 100×100 — десять тысяч измерений, а ковариационная матрица 10 000×10 000 для рабочих станций начала 1990-х была слишком ресурсоемкой. Для решения проблемы был использован «трюк со снимками» (*method of snapshots*): вместо огромной матрицы по числу пикселей разлагают маленькую матрицу по числу кадров в обучающем наборе, а найденные собственные векторы «поднимают» обратно в исходное пространство. Идею этого подхода Сирович предложил в 1987 для задач турбулентности, а вместе с Кирби в том же году перенес ее на лица. В результате оказывается достаточно нескольких десятков компонент, чтобы и хорошо восстанавливать изображение, и надежно отличать одно лицо от другого. Формальные метрики типа ошибок ложного допуска/ложного отказа и ТОП-1 идентификации станут стандартом чуть позже, с программой FERET (технология распознавания лиц) середины 1990-х, которая специально ввела единые протоколы тестирования и отчетности. До этого каждая группа публиковала результаты на своих наборах и с собственными процедурами; надежность фактически означала «высокий процент на своем датасете (наборе обучающих данных) плюс предсказуемое поведение при малых изменениях плюс скорость».

В итоге задачу надежности Терк и Пентленд решали комбинацией нескольких подходов: применяли потоковую локализацию головы в кадре, далее — проекцию найденного лица в «пространство лиц», затем сравнение короткого вектора весов с эталонами. На используемом тогда железе это работало быстро именно потому, что «тяжелая» математика была заранее сведена к малой размерности.

Method of snapshots

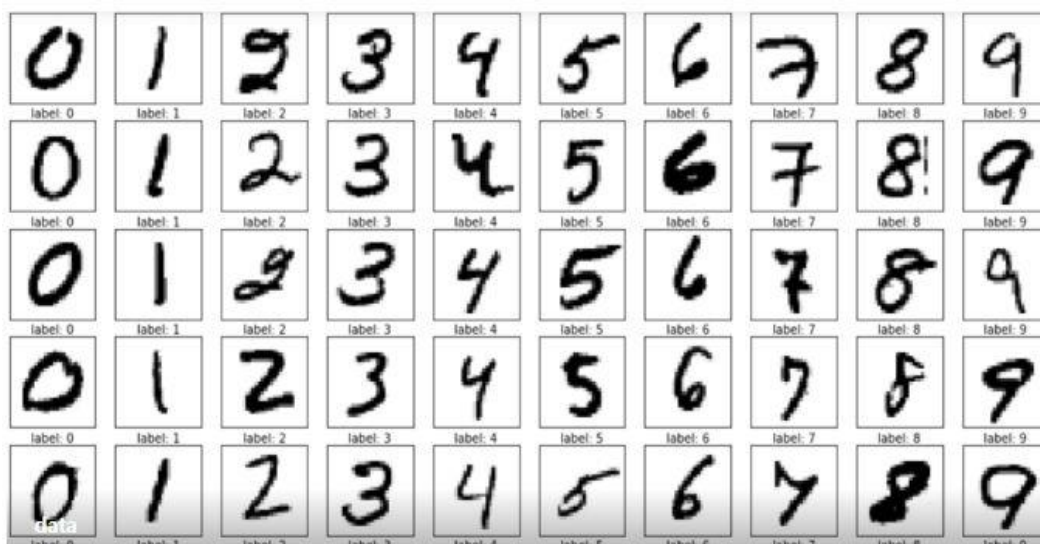
«**Method of snapshots**» Сировича (1987) считается классической, вычислительно эффективной формой «трюка со снимками», откуда он и пришел в машинное зрение. Лоуренс Сирович и Майкл Кирби, показали, что несколько десятков компонентов уже дают хорошую реконструкцию человеческих лиц; Терк и Пентленд перенесли этот подход в полноценный метод распознавания и впервые продемонстрировали «почти реальное время». Слабое место тоже было понятно уже тогда: освещение и повороты заметно меняют веса, и потому простая проекция в «пространство лиц» чувствительна к свету и позе — ограничение, которое в те годы пытались «лечить» более сложными линейными проекциями и нормировками.

Параллельно в Швеции и Нидерландах формировалось то, что сегодня называется «теорией масштабирующего пространства» — строгая математика про то, как *правильно* смотреть на изображение на разных

масштабах. Идея зародилась у Эндрю Уиткина в 1983 году: если сглаживать картинку гауссовым фильтром все крупнее и крупнее, мы получаем «стопку» все более грубых версий сцены без появления искусственных деталей — так масштаб начинает подчиняться понятным законам. В начале 1990-х Тони Линдерберг довел это до монографии и серии работ: при естественных требованиях к инвариантности масштаба единственно правильное сглаживание — именно гауссово, а его производные дают канонические «детекторы» черт на разных уровнях. Это звучит абстрактно, но последствия были значительные: стало возможным, не угадывая правильный размер окна, находить углы, «пятна» и другие особенности изображения там, где им положено быть при смене масштаба, — другими словами, автоматически выбирать подходящий масштаб для конкретной детали.

OCR

Следующая быстро развивающаяся в те времена технология, — **OCR (Optical Character Recognition)** - распознавание рукописных и печатных символов. В 1998 году группа легендарного Янна Лекуна опубликовала обзорную работу в *Proceedings of the IEEE* и представила то, что вскоре станет «Hello, world!» для любых алгоритмов машинного зрения: набор MNIST — 70 тысяч черно-белых картинок 28×28 с рукописными цифрами, разделенных на 60 тысяч для обучения и 10 тысяч для теста. Гениальность MNIST была в простоте: общедоступный, чистый, стандартизованный датасет, на котором можно сравнивать методы чисто и воспроизводимо. В той же статье Лекун и соавторы описали полный промышленный конвейер распознавания документов и показали, что сверточные нейросети не просто идея — они превосходят альтернативы на реальной задаче. А еще авторы зафиксировали стандарт, к которому мы привыкли и сегодня: цифра сначала нормализуется и укладывается в 28×28 , а затем подается в сеть — эта пре-обработка делает сравнения методов корректными.



Речь идет о промышленной системе чтения банковских чеков, которую в 1990-е сделала в AT&T Bell Labs (в связке с NCR Corporation) команда Яна Лекуна — вместе с Леоном Ботту, Йошуа Бенджио и Патриком Хаффнером. В ее «сердце» были две вещи: сверточные сети, распознающие отдельные символы и фрагменты рукописи, и графовая трансформирующая сеть (**Graph Transformer Network GTN**) — тренируемая «склейка» из модулей, которая навешивала на эти локальные распознавания **глобальные ограничения документа**. Использовалась элегантная инженерная идея: чек — это не просто картинка с цифрами, у него есть грамматика и согласованность полей. Система строила граф возможных сегментаций и интерпретаций, пропускала его через слои трансформаторов, где одни модули добавляли в граф «оценки» от сверточной нейросети, а другие — проверяли синтаксис суммы (как она должна выглядеть в цифрах и словах), соответствие между «courtesy amount» (цифрами) и «legal amount» (прописью), а также лексические и контекстные правила. Важный штрих: весь этот конвейер обучался **глобально по единой функции потерь** — градиент прокатывался через граф целиком, так что локальные распознаватели учились работать, чтобы в итоге правильно читался **весь** чек, а не только отдельная цифра. Это и имеется в виду под «графовой трансформирующей сетью»: модули принимают графы на вход, порождают графы на выход, и вся система оптимизируется как одно целое, то есть имеет сквозное определение.

Конкретно для сумм на чеках команда описала «грамматику денег»: что такое корректная запись доллара и центов, какие последовательности слов считаются валидными, как они должны совпадать с цифрами. Нейросеть делала распознавание символов и фрагментов, а GTN собирала из этого единственную согласованную трактовку. Техническая публикация с демонстрацией чтения чеков вышла в 1997 году (ICASSP),

а большой обзор — в *Proceedings of the IEEE* в 1998; там эта архитектура и зафиксирована как общий способ «глобальной тренировки документных систем» (сам термин Graph Transformer Networks — оттуда).

По меркам той эпохи она было не только инновационной, но и промышленно масштабируемой: к концу 1990-х такая система реально стояла в больших чек-ридерах и, по словам Лекуна и соавторов, читала **свыше 10%** всех чеков в США. Это яркий для своих лет пример массового внедрения нейросетевого машинного зрения и ранний прототип того, что мы сегодня называем сквозной системой распознавания документов.

В России именно документооборот сделал компьютерное зрение массовым. **ABBYY FineReader 1.0** вышел в 1993 году и с первых версий позиционировался как *omnifont*-распознавание — не под конкретный шрифт, а «поверх» разнообразных гарнитур на кириллице и латинице. Это был редкий пример коммерческого продукта по машинному зрению, который быстро обрел большой рынок, а позже стал де-факто стандартом OCR в нашем регионе.

Параллельно Когнитивные Технологии в 1993 выпускает **CuneiForm** (англ. *cuneiform*, «кьюниформ» — клинопись) — система оптического распознавания текстов: в 1993 библиотека распознавания попадает в CorelDRAW 3.0, а в 1994 Когнитивные Технологии заключает контракт с HP на поставку сканеров для России с предустановленным CuneiForm — первый такой договор HP с российской софтверной фирмой.

Когнитивные Технологии — это отдельное явление в истории российского машинного зрения, это одна из тех компаний, через которые хорошо видно, как в России собирался прикладной ИИ. Компания выросла из среды советской и постсоветской школы искусственного интеллекта, в том числе из круга, связанного с Лабораторией искусственного интеллекта Института системного анализа и командой, работавшей над шахматной программой «Каисса».

Первый большой сюжет в её истории — это, конечно, OCR, оптическое распознавание символов, о котором я уже упоминал ранее.

Дальше эта линия только расширялась. В открытых биографических и продуктовых справках перечисляются OEM-соглашения и интеграции с Brother, Canon, Xerox, Epson, Samsung, Olivetti, IBM и другими компаниями. Когнитивные Технологии оказалась среди тех, кто делал эту функцию как поставляемую библиотеку, которую можно было встроить в сканер, МФУ или издательский пакет. Позже исходные коды CuneiForm были открыты, и сам проект вошёл в историю уже не только

как коммерческий OCR, но и как одна из заметных открытых систем распознавания текста, выросших в России.

Для истории машинного зрения особенно важно другое: Когнитивные Технологии довольно рано закрепила в России представление о распознавании как о промышленном конвейере, а не только о наборе алгоритмов. Это означает, что компания работала не просто с моделью или библиотекой, а со всей цепочкой: сканер, изображение, предобработка, распознавание, интеграция в пользовательский продукт, поддержка языков, устойчивость к качеству печати, согласование с оборудованием партнёра. В каком-то смысле это был очень практически ориентированный подход, но с академической инженерной базой. Именно такой подход потом и позволила ряду российских компаний, вышедших из похожей школы, сравнительно быстро перейти к дорожному машинному зрению, промышленной видеоаналитике и автономной технике.

Позже компания стала заметна уже в совершенно другом контуре — в автономном транспорте и системах помощи водителю. В публичных справках Когнитивные Технологии связывают с проектом беспилотного КАМАЗа, стартовавшим в 2015 году; позднее из этой линии выросла более специализированная структура Когнитивный Пилот, которая уже работала на стыке компьютерного зрения, сельхозтехники и автономного управления. На сегодняшнем официальном сайте «Когнитив» компания вообще позиционирует себя прежде всего как российского разработчика программного обеспечения и прикладных решений для сельского хозяйства, в том числе автопилота для трактора с камерами, GNSS-навигацией, IMU и обработкой сенсорных данных на борту.

Отдельно стоит сказать о людях. Ольга Ускова в открытых биографиях проходит как основатель и публичное лицо компании, человек, который соединял исследовательскую среду и предпринимательскую практику. Владимир Арлазаров — фигура другого типа: представитель старшей научной школы, связанной и с «Каиссой», и с распознаванием образов ещё советского периода. Именно такое сочетание — предпринимательская энергия и сильная исследовательская база — во многом и объясняет, почему Когнитивные Технологии пережили несколько смен технологических эпох и прошли путь длинной эволюции: от OCR 1990-х к автономным системам 2010-х и 2020-х. Развитие машинного зрения — это не череда революционных прорывов, а постепенное наращивание класса задач, в которых система должна не просто распознавать, а действовать в реальном мире.

AdaBoost

Алгоритм AdaBoost, сокращение от Adaptive Boosting, представляет собой один из фундаментальных методов ансамблирования в машинном обучении, разработанный Йоавом Фройндом и Робертом Шапире в 1995 году. Он предназначен для преобразования множества слабых классификаторов — моделей, которые лишь немного лучше случайного угадывания, — в один сильный, с высокой точностью предсказаний. Основная идея заключается в итеративном обучении: на каждом шаге алгоритм фокусируется на объектах, которые были плохо классифицированы предыдущими моделями, корректируя их веса, чтобы последующие классификаторы уделяли им больше внимания.

Именно поэтому AdaBoost оказался особенно полезен в компьютерном зрении начала 2000-х. Самый известный пример — детектор лиц Виолы–Джонса 2001 года. Там использовались очень простые прямоугольные признаки Хаара, которых были десятки и сотни тысяч возможных вариантов, но только малая часть из них была действительно полезна. AdaBoost выбирал из этого огромного набора лучшие слабые классификаторы и объединял их в рабочий детектор. Более того, он хорошо сочетался с каскадной архитектурой: первые стадии быстро отбрасывали почти весь фон, а более поздние тратили вычисления только на трудные окна изображения. Именно эта связка — простые признаки, AdaBoost и каскад — сделала возможным детектирование лиц в реальном времени на тогдашнем бытовом железе.

Но лица — не единственное применение. В своё время AdaBoost широко использовали для детекции пешеходов, дорожных знаков, номерных знаков, для задач фильтрации спама, кредитного скоринга, медицинской диагностики, то есть везде, где было много простых признаков и нужна была интерпретируемая, быстрая и достаточно точная система. В более широком смысле AdaBoost стал важной вехой ещё и потому, что закрепил саму логику ансамблирования: слабые правила можно собирать в сильную модель, если делать это не хаотично, а с учётом ошибок предыдущих шагов.

Чтобы понять, как это работает, рассмотрим процесс шаг за шагом. Предположим, у нас есть обучающий набор данных с метками классов. Изначально все объекты имеют равные веса, суммирующиеся до

единицы. На первой итерации обучается базовый слабый классификатор, например, дерево решений глубиной единица. Затем вычисляется его ошибка взвешенной классификации: ϵ = сумма весов ошибочно классифицированных объектов. Вес классификатора рассчитывается так, чтобы дать больший вклад моделям с низкой ошибкой. Далее веса объектов обновляются: для правильно классифицированных они уменьшаются по экспоненте, а для ошибочных — увеличиваются, нормализуясь для следующей итерации. Этот цикл повторяется столько раз, сколько имеется базовых моделей. Финальное предсказание — бинарная классификация на основе суммы весов.

Такая адаптивная схема делает AdaBoost устойчивым к переобучению и эффективным для задач классификации, включая распознавание образов и биоинформатику. Однако он чувствителен к выбросам и шуму в данных, поскольку усиливает их влияние. В практике AdaBoost часто реализуется в библиотеках вроде `scikit-learn`, где базовые модели — это стампы или другие простые алгоритмы. Алгоритм остается актуальным, хотя и уступает современным вариантам бустинга, таким как XGBoost, в скорости и масштабируемости.

В компьютерном зрении это по-прежнему работает в простых детекторах на встраиваемых устройствах. Например, если нужно быстро и дешево выделять лицо, глаз, номерной знак, дорожный знак, силуэт пешехода или отдельные детали на конвейере, а вычислительный бюджет минимален. Если задача хорошо описывается набором простых признаков — контрастами, локальными границами, яркостными разностями, геометрией прямоугольных областей, — AdaBoost остаётся практичным вариантом. Особенно там, где картинка достаточно однотипна: фиксированная камера, постоянный ракурс, стабильное освещение.

В промышленной автоматизации он удобен для визуального контроля брака. Камера смотрит на одну и ту же деталь, а признаки можно задать сравнительно простые: наличие отверстия, правильность кромки, контраст штампа, полнота сборки. Если дефектов немного, а примеров мало, AdaBoost часто оказывается быстрее в разработке и понятнее в эксплуатации, чем нейросеть. То же относится к сортировке продукции, контролю упаковки, проверке маркировки и обнаружению посторонних предметов на линии.

В видеоаналитике и системах безопасности он полезен для предварительной фильтрации потока. Например, можно быстро отсекать заведомо пустые кадры, грубо детектировать человека в зоне, выделять движение нужного типа, отбрасывать шумы от бликов и теней. То есть использовать его как лёгкий первый каскад перед более дорогой моделью. Это особенно полезно, когда камер много, а ресурсы на сервере ограничены.

В медицинских и биометрических задачах AdaBoost тоже всё ещё встречается там, где данные компактные, а признаки извлечены заранее. Например, для классификации по лабораторным показателям, для оценки риска по небольшому набору измерений, для предварительного отбора подозрительных случаев на дообследование. Его сила здесь в том, что он хорошо собирает множество слабых сигналов в одно итоговое решение.

В табличных данных он вообще остаётся очень эффективным. Кредитный скоринг, антифрод, прогноз отказов оборудования, оценка оттока клиентов, фильтрация заявок, диагностика по сенсорным данным — везде, где у нас есть набор числовых или бинарных признаков и хочется получить сильную, но не слишком тяжёлую модель, AdaBoost всё ещё уместен. Хотя сегодня в этой области его часто теснят более современные градиентные бустинги вроде XGBoost, LightGBM и CatBoost, сама логика остаётся той же: строим сильную модель из последовательности простых решений. Они лучше переносят разнородные признаки, пропуски, нелинейные взаимодействия и обычно дают более спокойный результат на реальных данных. Если же вам нужна максимально прозрачная и калиброванная модель, можно начать ещё проще — с логистической регрессии или неглубокого дерева решений, а уже потом посмотреть, даёт ли бустинг прирост.

В моем бильярдном проекте я использую этот подход для принятия решения о результате удара по нескольким слабым и средним источникам истины, каждый из которых по отдельности ошибается, но делает это по-разному. А значит, их можно собрать в единый ансамбль и научить модель выдавать итоговую вероятность того, что шар действительно был забит.

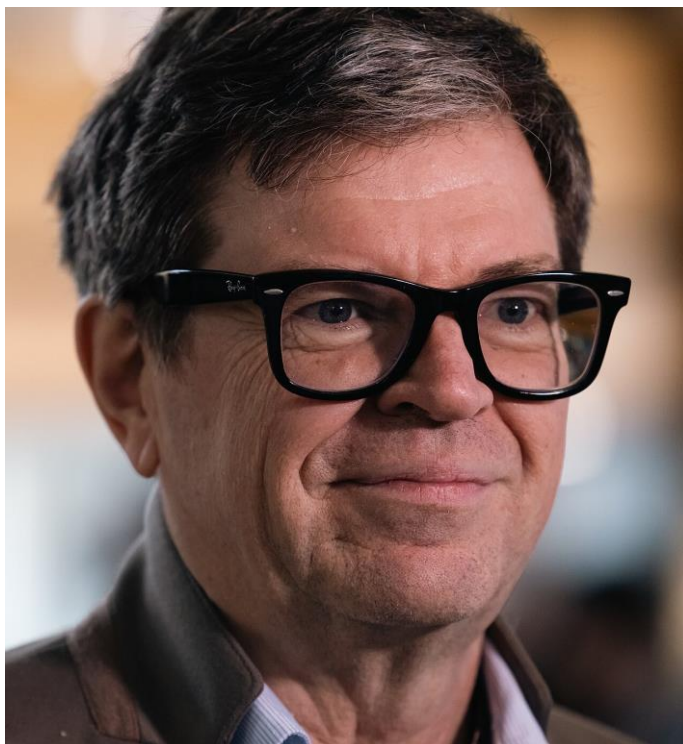
На входе есть несколько переменных, сколько шаров было на столе до удара и после, уверенность нейросетевого детектора лузы, факт исчезновения траектории вблизи конкретной лузы, длительность пропадания шара, наличие движения в зоне лузы, согласованность с физической симуляцией, уверенность трекара, был ли перекрыт шар рукой или кием, не появился ли он снова через несколько кадров, насколько правдоподобна итоговая динамика остальных шаров. Всё это хорошие кандидаты на признаки для ансамбля.

AdaBoost в такой схеме работает как надстройка над этими сигналами. Он последовательно учится на ошибках: если, например, простое правило «шаров стало меньше на один» часто ломается на перекрытиях или на ложных пропажах трека, то следующие слабые классификаторы начнут сильнее опираться на другие признаки — на геометрию траектории, на локализацию около лузы, на симуляцию или на уверенность нейросети. То есть итоговая модель получится не «средним

арифметическим источников», а системой, которая учится, в каких ситуациях какому источнику больше верить.

Но здесь есть практическая тонкость. AdaBoost особенно хорош, когда признаки относительно простые, а слабые правила можно выразить в виде порогов или маленьких деревьев - «если шар исчез в радиусе R от лузы и не появился N кадров, это слабый сигнал в пользу забития»; «если количество шаров на столе уменьшилось, это другой слабый сигнал»; «если симуляция подтверждает эпизод, это третий сигнал». Вот это называется классической бустинговой задачей.

LeNet



Есть одна выдающаяся фигура, благодаря которой машинное зрение вышло из эпохи запрограммированных признаков к эре машинного обучения, это Ян Лекун. Этот человек стоял у истоков архитектуры сверточных нейросетей, и много лет последовательно отстаивает универсальный принцип: система машинного зрения должна не получать заранее придуманный набор признаков, а учиться строить собственные представления из данных. Именно с этой позицией связаны и его ранние работы по рукописному тексту, и LeNet, и более поздняя защита самоконтролируемого обучения, где машина извлекает структуру мира не из бесконечных ручных меток, а из самого потока наблюдений. За фундаментальные концептуальные и инженерные прорывы в глубоком обучении Лекун вместе с Джеффри Хинтоном и Йошуа Бенджио получил премию Тьюринга за 2018 год.

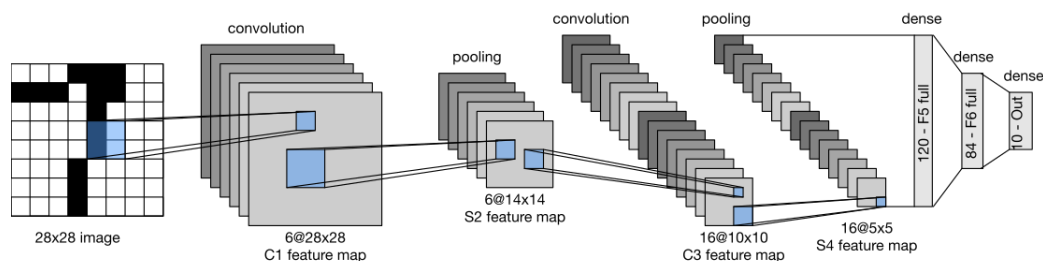
Лекун родился 8 июля 1960 года и вырос во Франции, в пригороде Парижа. Его отец был инженером, и интерес к электронике и компьютерам перешёл к сыну ещё в детстве. В юности он не сводился к одной только технике: любил музыку и играл в группе, но при этом рано занялся инженерными задачами и машинным интеллектом. Учился он в Высшей школе инженеров в области электротехники и электроники в Париже (ESIEE Paris), затем защитил докторскую работу о том, как обучать многослойные нейросетевые системы, включая ранние варианты алгоритма обратного распространения ошибки, и как сделать такие сети пригодными для реальных задач распознавания в Университете Пьера и Марии Кюри, после чего на короткое время попал в Торонто к Хинтону, а в 1988 году перешёл в Bell Labs. Именно там его группа сделала одну из первых, реально внедрённых нейросетевых систем оптического распознавания символов для банковских чеков, о которой я писал ранее.

Отдельно стоит сказать о его исследовательской философии. Лекун много лет последовательно спорит с идеей, что интеллект можно построить только на языке и только на масштабировании текстовых моделей. Его позиция другая: машинная система должна учиться наблюдать мир, отделять существенное от случайного и строить внутреннюю модель причинной структуры среды. В современных работах и выступлениях он снова и снова возвращается к одной мысли: обучение должно быть в значительной степени самоконтролируемым, то есть идти через наблюдение и предсказание, а не только через внешние ярлыки. В формулировке, которую хорошо передаёт недавний текст NYU, интеллект для него связан не просто с накоплением ответов, а с умением научиться игнорировать нерелевантное и удерживать то, что действительно важно для прогноза и действия.

Сейчас он по-прежнему отстаивает ту же позицию, только на новом уровне: настаивает, что следующая ступень ИИ должна учиться не только классифицировать и продолжать тексты, но и понимать физический мир, время, действия и последствия. В этом смысле Лекун заметен не только как автор конкретных архитектур, но и как человек, который постоянно возвращает дискуссию к фундаментальному вопросу: что именно должно быть выучено внутри системы, чтобы она действительно понимала сцену, а не просто угадывала ответ по статистике.

В 1998 он совместно с коллегами в той же Bell Labs, на практике разработал LeNet-5. До неё уже существовали более ранние версии (LeNet-1, LeNet-4), в которых были заложены основные принципы — локальные фильтры, обучение через обратное распространение ошибки, попытка уйти от ручного конструирования признаков. Но именно LeNet-5 стала первой системой, где всё это было собрано в целостную, воспроизводимую архитектуру под конкретную задачу — распознавание рукописных цифр на чеках и почтовых индексах.

Важно понимать контекст. В то время компьютерное зрение в основном опиралось на вручную заданные признаки: границы, углы, гистограммы. Это работало, но требовало постоянной подстройки под задачу. Идея, что система сама научится извлекать устойчивые признаки из сырых пикселей, выглядела неочевидной и во многом рискованной. LeNet-5 убедительно доказала, что это возможно на практике: если правильно задать структуру сети и процедуру обучения, модель сама находит те элементы изображения, которые действительно важны.



Архитектура LeNet-5 построена как последовательность чередующихся слоёв: свёртки, затем уменьшение разрешения (пулинг), снова свёртки и в конце несколько полносвязных слоёв для классификации. На вход подаётся изображение фиксированного размера, дальше сеть постепенно превращает его в набор всё более абстрактных представлений. Сначала выделяются простые элементы — линии, изгибы, контрасты. Затем из них собираются более сложные структуры — части цифр. На верхнем уровне остаётся компактное представление, из которого уже легко принять решение о классе.

Смысл свёртки можно объяснить просто: это способ искать один и тот же признак по всему изображению. Один фильтр «сканирует» картинку и реагирует, например, на вертикальные штрихи. Благодаря тому, что этот фильтр общий для всех позиций, сеть не привязывается к конкретному месту — она учится самому признаку. Это резко сокращает число параметров и делает систему устойчивой к сдвигам.

Пулинг выполняет другую роль — он снижает чувствительность к мелким изменениям. Если признак немного сместился или исказился, его отклик в соседней области всё равно сохранится. В LeNet-5 использовалось сглаживающее уменьшение разрешения, а не жёсткий максимум, что дополнительно стабилизировало представление. В результате сеть постепенно теряет точную информацию о координатах, но сохраняет устойчивые различия, которые важны для распознавания.

Отдельная инженерная деталь — частичная связность между слоями. Не все карты признаков соединяются со всеми следующими слоями. Это уменьшает число параметров и заставляет сеть формировать более разнообразные признаки, а не дублировать одни и те же. Для своего времени это было важным компромиссом между выразительностью и вычислительными ограничениями.

На уровне здравого смысла свёртка отвечает на вопрос: «что здесь за локальный признак?», а пулинг — на вопрос: «насколько этот

признак устойчиво присутствует в этой области?». Свёртка уточняет, пулинг огрубляет. Свёртка чувствительна, пулинг терпим. Свёртка находит, пулинг закрепляет.

Именно поэтому они так хорошо работают в паре. Если оставить только свёртки, сеть будет слишком зависеть от мелких колебаний: линия чуть сдвинулась, блик ушёл, контур стал толще — и карта признаков уже изменилась. Если оставить только пулинг, мы потеряем слишком много структуры и всё станет чрезмерно грубым. Вместе они создают ритм: сначала сеть внимательно смотрит, потом делает шаг назад и обобщает увиденное. Потом снова внимательно смотрит уже на новом уровне, потом снова обобщает.

На примере бильярда это особенно легко почувствовать. Первый сверточный слой замечает светлую дугу блика, тёмную кромку шара, границу между сукном и бортом. Пулинг после этого позволяет адаптировать этот результат к разным ситуациям, если шар в следующем кадре сдвинулся или его контур чуть размазался. Следующий сверточный слой, более крупный, уже начинает «понимать», что комбинация круглой границы, блика и локальной тени — это скорее шар, а не просто случайное пятно. Ещё выше сеть различает, где шар стоит отдельно, а где прилип к борту или к другому шару. И на каждом уровне пулинг помогает сохранить не точную геометрию пикселей, а устойчивый факт присутствия нужной структуры в разных условиях.

Свёртка похожа на работу узкого специалиста, который решает свою конкретную задачу, а пулинг — на работу начальника, которому не нужны все мелочи, ему нужно краткое понимание кто чем занимается. Нижние слои дают детали, верхние начинают мыслить уже в более крупных категориях. Если переборщить с пулингом, сеть станет «близорукой» к тонким границам и потеряет точность локализации. Поэтому в задачах сегментации и детекции потом придумали архитектуры, где часть высокодетальной информации возвращают обратно. Но сам базовый принцип остаётся прежним: сначала найти, потом обобщить.

Если посмотреть на LeNet-5 чуть шире, её главное достижение — в подтверждении правильности выбранного пути. Она показала, что изображение удобнее обрабатывать как иерархию: от локальных простых признаков к глобальному решению, причём все уровни обучаются совместно. Эта идея оказалась устойчивой. Практически все последующие архитектуры — от более глубоких сверточных сетей до современных гибридных моделей — так или иначе продолжают эту логику. Именно после неё стало понятно, что отказ от ручных признаков — это рабочее направление для более универсальных задач.

SIFT

И наконец, 1999 год, алгоритм SIFT (scale-invariant feature transform) расшифровывается как «масштабно-инвариантное преобразование признаков» появился в технологиях компьютерного зрения. Дэвид Лоу показал, как извлекать из изображения локальные особенности, практически не боящиеся ни увеличения, ни поворота, ни даже умеренных изменений освещения и перспективы. Алгоритм искал «ключевые точки» как экстремумы в гауссовой «стопке масштабов», а вокруг каждой строил дескриптор — компактный «отпечаток» градиентов, привязанный к собственной шкале и ориентации. В демонстрациях того времени это выглядело почти как магия: взять фотографию памятника, повернуть, масштабировать, даже частично закрыть его — а SIFT по нескольким десяткам совпавших «ключей» все равно уверенно находит соответствия и оценивает преобразование сцены. Что особенно впечатляло, на рабочей станции Sun SPARC station 10 построение пирамиды и ключей занимало порядка девяти десятых секунды, а полная схема сопоставления и верификации укладывалась менее чем в две секунды на изображение. Для конца 1990-х это означало не только устойчивость, но и практичность — можно собирать панорамы, сопоставлять кадры, искать объекты в каталоге.

В начале десятилетия исследователи хотели *сжать* изображение до смыслового ядра — эйгенфейсы отвечали именно на это: как хранить и сравнивать лица, не утонув в тысячах пикселей. Но выяснилось, что мир меняется с масштабом, и без правильной работы по шкалам любой детектор превращается в гадание — тут на сцену вышла теория масштаб-пространства. Железо все еще сильно ограничивало: рабочие станции Sun и SGI, десятки мегабайт памяти, тактовые частоты в сотни мегагерц. Поэтому архитектуры и алгоритмы проектировали «с оглядкой»: вычисления — локальные и иерархические, сравнения — в компактных пространствах, устойчивость — встроенная, а не натянутая постфактум. Именно эти компромиссы и сделали девяностые такой плодотворной эпохой: на их основе стало возможно и массовое тестирование (MNIST), и выделение ключевых точек из шумных реальных сцен (SIFT), и первые промышленные нейросетевые алгоритмы (чтение чеков). В совокупности все это подготовило почву для следующего этапа — когда данные станут огромными, GPU подешевеют, а «глубина» нейросетей вырастет на порядки.

ЧАСТЬ 2

Распознавание образов

Начало тысячелетия у сообщества энтузиастов машинного зрения было символично увенчано появлением «общего языка» — библиотеки, которая облегчает разработку и дает всем одинаково быстрые, проверенные пазлы для решения многочисленных задач машинного зрения. 13–15 июня 2000 года в **Hilton Head Island, Южная Каролина (США)** на ежегодной конференции IEEE (*Institute of Electrical and Electronics Engineers*) некоммерческая ассоциация инженеров и исследователей, чья миссия — развивать технологии «на благо человечества», по компьютерному зрению и распознаванию образов, **CVPR-2000**, корпорация Intel официально представила публике первую альфа-версию OpenCV — открытой библиотеки функций для компьютерного зрения. Мы шагнули в эпоху общего, бесплатного и при этом производительного кода. Затем последовали несколько бета-релизов, а стабильная 1.0 вышла в 2006.

Обратите внимание на миссию этой некоммерческой ассоциации инженеров и исследователей - развивать технологии на благо человечества. Спустя 25 лет вопрос, насколько эти технологии и корпорации, которые за ними стоят, послужили во благо человечеству, остается неоднозначным. С одной стороны, усиление централизованного контроля во многих аспектах личной жизни, ограничение свободы выбора, расслоение общества, вред физическому и психологическому здоровью целого поколения. С другой стороны, автоматизация рутинных задач, новые перспективы для творческой самореализации.

Вовлечение инженеров и изобретателей в широкую дискуссию, касательно морально этических аспектов развития технологий и повышения уровня самосознания специалистов, я считаю для себя самой основополагающей задачей из всех своих проектов. С этой целью в 2025 году в Крыму я учредил прототип будущей некоммерческой ассоциации «Кремниевое Братство». Это место встречи единомышленников, расположенное в самом живописном месте южного берега Крыма, куда можно круглогодично приехать с целью обмена мнениями, участия в дискуссиях и продвижения морально этических взглядов среди инженеров, изобретателей и энтузиастов технического творчества, что естественным образом содействует повышению социального и общекультурного самосознания общества. Вместе с этим, и как его следствие, достигается сопутствующее повышение качества и уровня производимых ими технологий. Это очевидный результат самодисциплины, честного взгляда на вещи, оптимизации и способности переходить от частных

к общим категориям, от практической эвристики к математическим моделям. Пользуясь возможностью, со страниц этой книги, я приглашаю всех заинтересованных читателей связаться со мной по электронной почте stuurgurs@yandex.ru для совместной работы в этом глобальном направлении и координации наших общих действий по повышению социального самосознания.

Возвращаясь в начало 2000х, посмотрим, как Intel запускала исследовательские проекты, нацеленные на тяжелые, ресурсоемкие задачи — от трассировки лучей в реальном времени до видеостен, — и хотела дать исследователям и инженерам оптимизированную платформу, продвигая свои процессоры в профессиональном сообществе. В официальной истории OpenCV это сформулировано предельно открыто: не просто бесплатный, а оптимизированный под процессоры Intel модульный код как общее инфраструктурное основание, портируемые, производительные блоки, которые можно бесплатно внедрять в коммерческие продукты.

С самого начала в разработку были вовлечены оптимизационные команды Intel в России и группа, делавшая производительные библиотечные ядра: под них переписывались базовые алгоритмы, от фильтрации до калибровки, под SIMD-инструкции и многоядерные процессоры. Вскоре после релиза альфы OpenCV, Intel официально объявляет о создании центра разработки в Нижнем Новгороде — именно там выросла та команда, которая на годы вперед будет собирать и выпускать библиотеку OpenCV. Техническим лидером разработки стал Вадим Писаревский, он возглавил нижегородскую группу еще в 2000 и много лет руководил архитектурой и интеграцией с производительными библиотеками Intel.

В ноябре 2000 Гэри Брадски публикует в Dr. Dobbs's Journal программную статью «The OpenCV Library» — по сути, первый публичный манифест библиотеки, где обещан именно реал-тайм: кроссплатформенная C/C++-база, готовые модули для `imgproc`, `calib3d`, видео и I/O, плюс проприетарные базы Intel, если они есть в системе.

Уже к середине 2000-х OpenCV использовалась в промышленных задачах. Знаковый эпизод: автономный автомобиль Stanford Stanley, победивший в DARPA Grand Challenge 2005, о котором речь пойдет дальше, в части 4 про автономный транспорт, использовал OpenCV в своем технологическом стеке и был демонстрацией того самого замысла «инфраструктуры для реального времени». В популярной обзорной статье ACM Queue «Realtime Computer Vision with OpenCV» (среди авторов — Анатолий Бакшеев, Кирилл Корняков, Виктор Ерухимов) заметная часть инженерной экспертизы по-прежнему идет из России, где

вокруг нижегородского центра Intel сформировалась устойчивая школа оптимизации CV-кода.

В 2010-е ключевые инженеры из Нижнего Новгорода выносят экспертизу в отдельную компанию Itseez, которая несколько лет фактически тянет активную разработку OpenCV и стандартизацию прикладных API (от OpenCV contrib до OpenVX), а в 2016 Intel приобретает Itseez, формально «замкнув» круг: развитие библиотеки возвращается туда, где начиналось, только уже с выросшей командой и зрелой экосистемой.

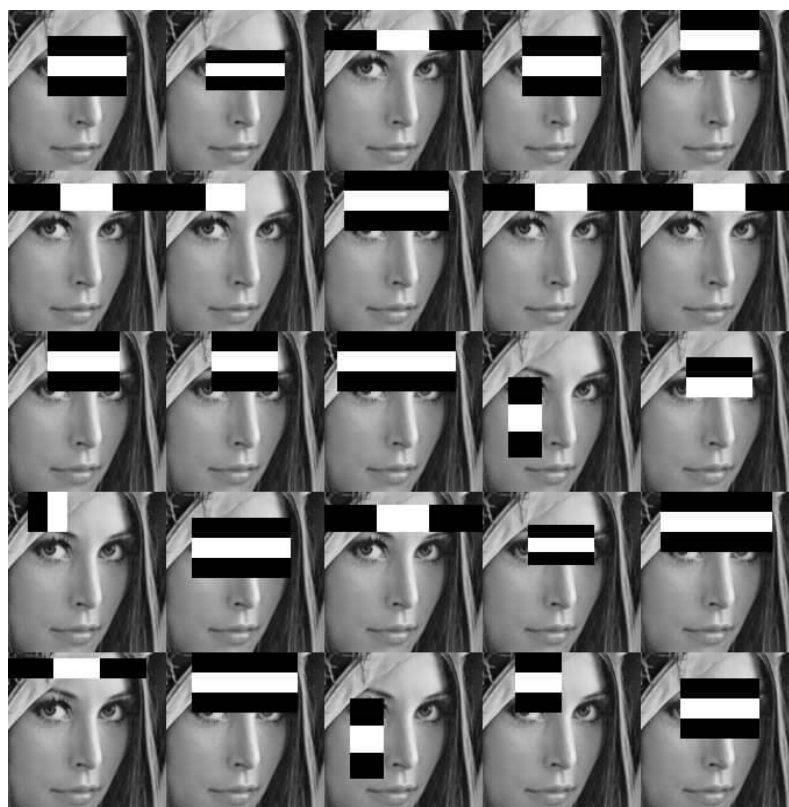


Лена Седерберг (Lena Söderberg Forsén), шведская модель и актриса с русским именем, стала невольной иконой цифровой обработки изображений благодаря портретному фото, снятом фотографом Дуайтом Хукером для журнала Playboy в ноябре 1972 года, где она была представлена как модель месяца. Обрезанная версия этого снимка — с ее лицом, плечом и частью шляпы — впервые вошла в историю в 1973

году, когда инженер Александр Савчук из Университета Южной Калифорнии использовал ее в своей статье о сжатии изображений, отметив высокое качество тестовой цели. Мы с вами, живые люди, прекрасно понимаем между строк, что на самом деле имелось ввиду под сухим определением «высокое качество тестовой цели». Вот и для остальных было очевидно почему изображение Лены превратилось в универсальный эталон для алгоритмов машинного зрения: от фильтрации шума и детекции краев до компрессии и восстановления, пройдя через тысячи публикаций и библиотек. В OpenCV фото появилось как стандартный тестовый файл `lena.png` в примерах кода для C++ и Python, демонстрируя базовые операции вроде свертков или трансформаций. Не удивительно, что случайный кадр из гламурного журнала стал фундаментом научного прогресса, напоминая, что даже в строгом мире алгоритмов человеческий образ остается неиссякаемым источником вдохновения. В марте 2024 года институт IEEE официально опубликовал уведомление (Editorial: Farewell, Lena), запрещающее использование этого изображения в будущих научных работах. Сам факт ограничения как следствия большой популярности картинке среди исследователей, наглядно свидетельствует о наличии в человеке внутренних ценностных ориентиров, которые в более широком смысле, со всеми своими многочисленными внутренними взаимосвязями и временными противоречиями, составляют фундамент морально нравственных и цивилизационных основ человеческого общества и всех его индивидов, в частности. Общий интеллект и человеческий разум, таким образом, это живая открытая система, далеко выходящая за рамки отдельно взятого мозга в какой-то конкретный момент времени. С этой точки зрения искусственный интеллект стоит рассматривать не как отдельное понятие, а как один из множества имеющихся инструментов и социально культурных факторов развития нашего общества. Нужно быть живым человеком чтобы хотеть и осознавать как и для чего использовать тот или иной инструмент. Живым значит погруженным в определенный пространственно-временной контекст. Если бы научную работу писал не Александр Савчук, а условно независимый, бездушный алгоритм, какую картинку бы он назвал «высоким качеством тестовой цели»? А следующий вопрос был бы какая цель его научной работы? Кто смог бы поруиться за её морально нравственную чистоту и общую безопасность?

Каскадная детекция Виолы–Джонса (Viola-Jones)

В 2001 году Пол Виола и Майкл Джонс перевели задачу обнаружения объектов на изображении в более практическую плоскость чем это было принято ранее: вместо того чтобы распознавать лицо в уже подготовленном и выровненном кадре, они предложили искать его с помощью скользящего окна по всему изображению. Для этого они отказались от сложных глобальных признаков на основе отдельных пикселей в пользу простых локальных шаблонов контраста — так называемых «Хаар-подобных» прямоугольных признаков, которые сравнивают среднюю яркость в соседних областях изображения, например, проверяя, светлее ли нос по сравнению с глазами. Смотри пример на изображении ниже. Ключевой математический трюк здесь — создание «интегрального изображения»: это специальная предварительная таблица, где для каждого пикселя заранее вычислены суммы яркостей всех пикселей слева и сверху от него, подобно префиксным суммам в одномерном массиве. Благодаря этому любая сумма пикселей в произвольном прямоугольнике теперь рассчитывается всего за несколько простых операций, независимо от размера области. В результате вычисление тысяч таких базовых признаков для каждого окна сдвига стало реальным даже на относительно слабых процессорах начала 2000-х, превратив задачу из вычислительно неподъемной в вполне осуществимую.



Они упростили классификацию, сделав ее дискриминативной и разреженной: вместо одной мощной, но сложной линейной трансформации, подобной той, что используется в методе главных компонент (РСА), они создали ансамбль из множества простых правил — это всего лишь пороговые проверки по отдельным признакам. Алгоритм AdaBoost из тысяч доступных признаков отбирает десятки или сотни наиболее эффективных, которые лучше всего разделяют изображения с лицами и без них, и объединяет их в единую сильную модель, минимизируя экспоненциальную функцию ошибки для повышения точности. Ключевой инженерный прием — каскадная структура: первые этажи каскада используют всего несколько признаков и настроены так, чтобы пропускать почти все настоящие лица, но сразу отвергать подавляющее большинство фоновых окон как ложные. Только редкие «кандидаты», прошедшие начальные фильтры, попадают на более сложные, ресурсоемкие этажи. С математической точки зрения, общая вероятность пропуска лица остается низкой, поскольку она рассчитывается как произведение высоких показателей обнаружения на каждом этапе, а общий уровень ложных срабатываний резко падает как произведение малых вероятностей ошибок по этапам — это позволяет обрабатывать миллионы окон изображения за секунду, сохраняя при этом приемлемую точность.

Изменение подхода затронуло саму основу признаков: вместо равномерного, глобального описания яркости по всему изображению перешли к набору простых локальных контрастов — разниц в яркости между соседними областями, — которые устойчивы к сдвигам изображения (трансляциям) и меньше зависят от изменений освещения, а вычислять их быстро и просто. Кроме того, полностью сменился тип обучения: от несупервизированной модели, которая просто сжимает данные в подпространство (как в методе главных компонент, РСА), к супервизированному бустингу, где алгоритм жестко настраивается на минимизацию именно ошибки обнаружения. Именно эта комбинация — интегральное изображение для быстрого расчета, бустинг для отбора признаков и каскад для поэтапной фильтрации — позволила добиться того, чего не умели ранние методы на основе «собственных лиц»: полноценного сканирования сцены в реальном времени на обычном процессоре начала 2000-х без графических ускорителей, с хорошей устойчивостью к фону, разным масштабам и умеренным колебаниям освещения или позы. И все это без специальных уловок, которые работают идеально на одной камере, но выходят из строя при смене оборудования. Никакого вычитания кадров, чтобы выделить движение лиц, никакого поиска по цвету кожи. Вместо этого — чисто автоматическая система: одно и то же скользящее окно, один набор базовых контрастных признаков, одна обученная модель, которая проходит по изображениям разных масштабов в пирамидальной структуре. Вот почему скорость в 15 кадров в секунду

стала настоящим прорывом: это не подогнанный под конкретную сцену трюк, а универсальный метод, легко адаптируемый к разным камерам и условиям. Спустя считанные годы XML-файлы с «фронтальными» и «профильными» каскадами для распознавания лиц в профиль и в фас станут частью OpenCV, а детекторы лиц — штатной функцией бытовых камер. В итоге появилась промышленная технология не только «идентификации» лица, но и его распознавания достаточно быстрого, чтобы следить за ним в потоке видео, фокусироваться в фотоаппарате, и прочих отраслях.

А теперь, чтобы перейти от теории к практике, приведу пример из моих ранних экспериментов с бильярдным проектом: именно Viola-Jones стал одним из первых инструментов, который я применил для обнаружения шаров на кадрах видео из трансляций матчей. Алгоритм уже тогда демонстрировал приемлемую точность — около 85–90 процентов правильных детекций, — хотя и не абсолютную, поскольку иногда путал шары с бликами на столе или тенями от кия, особенно при неидеальном освещении. Некоторые шары не находил, особенно в сложных ситуациях. По сегодняшним меркам его скорость поражала: обработка полного кадра занимала доли секунды исключительно на процессоре. Для настройки я обучил собственный классификатор на обширной выборке — тысячах положительных примеров с изображениями шаров и аналогичном количестве отрицательных фрагментов без них, — все образцы были вручную размеченными и вырезанными с помощью утилиты `createsamples` из OpenCV, которая генерирует вариации на основе базовых фото. Обучение через инструмент `traincascade` заняло всего пару часов на обычном CPU, без изнуряющих экспериментов с параметрами; я остановился на стандартных оптимальных значениях, проверенных сообществом: размер окна детекции ширина и высота по 20–24 пикселя, чтобы захватывать шары среднего диаметра; `featureType` как HAAR для контрастных локальных признаков; `stageType` BOOST с `numStages` в 20 этапов для баланса между скоростью и точностью; `minHitRate` 0,95, чтобы минимизировать пропуски настоящих шаров, и `maxFalseAlarmRate` 0,5 на начальных стадиях с постепенным снижением до 10^{-6} на финальных, обеспечивая низкий уровень ложных срабатываний; препроцессинг с весами образцов `weight` 1,0 и без сильной аугментации, чтобы избежать переобучения.

Алгоритм Viola-Jones (метод каскадной детекции объектов на основе AdaBoost и Haar-подобных признаков) применялся в российских системах автоматической фиксации нарушений ПДД для локализации и распознавания автомобильных номеров, особенно в ранних этапах развития технологий (середина 2000-х — начало 2010-х годов). Пилотные комплексы, которые одновременно измеряли скорость и пытались рас-

познавать номер, начали испытывать еще в 1990-е, но массовая установка систем автоматической фотовидеофиксации пошла после появления правовой базы примерно к 2008 году: сначала Москва, потом большинство регионов, и к 2025 на дорогах уже десятки тысяч камер. В начале 2000-х алгоритмический «скелет» всех таких систем был очень похож: локализовать машину и область номерного знака, нормализовать изображение (перспектива, яркость), сегментировать символы и распознать их, плюс замерить скорость (радар или видео) и понять тип нарушения. Для этого использовали классические методы: пороговую обработку, поиск прямоугольных областей и линий методом Хафа, анализ гистограмм яркости, связные компоненты, а для распознавания — статистические классификаторы вроде SVM.

На этом фоне метод Виолы–Джонса занял очень практичную нишу: его начали переучивать на прямоугольные объекты — номерные знаки, силуэты машин, области интереса в кадре. Уже к середине 2010-х, когда вычислительная мощность подешевела, а архивы дорожного видео выросли, российские разработчики комплексов вроде «АвтоУрагана» и других стали переходить на сверточные нейросети и детекторы типа YOLO/SSD для машин и номеров, а также специализированные CNN/CRNN-модели для распознавания символов, но даже современные обзоры все еще упоминают связку «Viola–Jones + Хаф + SVM» как рабочий вариант там, где ресурсы сильно ограничены — в дешевых и встроенных системах.

Если говорить об «АвтоУрагане» как о заметной российской системе машинного зрения на дороге, то интересен он не только как коммерческий продукт, но и как хороший пример всей эволюции отрасли. За этим названием стоит компания «Технологии Распознавания», основанная в 2006 году; на её официальном сайте прямо сказано, что команда изначально строила многофункциональную систему распознавания регистрационных знаков и фиксации нарушений ПДД, а сам «АвтоУраган» стал одним из её ключевых аппаратно-программных комплексов.

По своей сути «АвтоУраган» — это не одна камера, а целая линейка комплексов и единое программное ядро. В публичных материалах фигурируют стационарный комплекс «АвтоУраган-ВСМ2», мобильные решения вроде «АвтоУраган-МС», а также связанное программное обеспечение «СПО АвтоУраган», которое производитель описывает как модульную клиент-серверную систему: её можно разносить по нескольким компьютерам или собирать локально на одном узле. Это важная деталь, потому что в таких системах сама «умность» давно уже не живёт внутри одной камеры: камера даёт поток и доказательный материал, а логика

распознавания, трекинга, сверки с базами и формирования события может жить на сервере или распределённо. Эта архитектура очень похожа на ту, что я применяю в бильярдном проекте PerformStars.com.

Базовая задача комплекса: распознать государственный регистрационный знак, привязать его ко времени, месту и направлению движения, измерить скорость или определить иной состав нарушения и сохранить доказательный материал. На официальной странице «АвтоУраган-ВСМ2» перечислены типовые нарушения, которые система умеет фиксировать автоматически: превышение скорости, проезд на красный, выезд за стоп-линию, выезд на встречную полосу, движение по обочине, по выделенной полосе, по трамвайным путям встречного направления, непредоставление преимущества пешеходу, нарушение запретов для грузового транспорта и ещё ряд сценариев. Для скоростного режима система, по данным производителя, умеет не только распознавать номер и измерять скорость, но и сохранять стоп-кадры, отдельное изображение номера и видеофрагмент с динамикой эпизода.

Здесь особенно интересно, как решены сами измерительные и логические задачи. Производитель отдельно подчёркивает, что комплекс умеет измерять скорость как по видеоизображению, так и радарным методом, в диапазоне до 350 км/ч, а в состав входит поверенный приёмник ГЛОНАСС/GPS. Там же сказано, что для контроля средней скорости на протяжённом участке используется общая синхронизация рубежей по спутниковому времени и предварительная градуировка по реальному профилю дороги — с учётом высот, изгибов и поворотов. Это важная инженерная деталь: в дорожной автоматике всё держится не только на распознавании картинки, но и на метрологии, синхронизации и привязке к геометрии трассы.

Видно, как отрасль постепенно уходила от «отдельной камеры для каждого сценария» к более вычислительной логике. В официальном буклете прямо описаны два метода: первый использует дополнительную камеру, второй — программный трекинг движения транспортного средства в зоне распознающей камеры. Производитель формулирует это довольно прямо: программа строит трек объекта по его смещению в зоне контроля и уже по изменению этого трекинга определяет, нарушил ли автомобиль разметку или запрет манёвра.

Ещё одна сильная сторона «АвтоУрагана» — работа не только как «штрафующей» камеры, но и как инструмента регистрации всего потока. В материале компании о розыске транспорта сказано, что каждый комплекс круглосуточно отправляет данные в единый центр обработки, где по каждой регистрации сохраняются дата, время, место установки, направление движения, распознанный или нераспознанный номер, а также дополнительные признаки вроде скорости или типа транспортного

средства. Там же упомянут довольно характерный модуль: автоматическое определение зоны лобового стекла с последующим «просветлением» изображения, чтобы лучше видеть, кто находится в салоне. С точки зрения машинного зрения это любопытно не само по себе, а как пример того, что дорожные комплексы давно работают не только с номером, но и с более широким визуальным описанием машины и сцены.

Особенно заметно это стало на задачах тяжёлого транспорта. В публичных материалах производителя говорится, что «АвтоУраган-ВСМ2-М» использовался в системе «Платон» и на рубежах контроля грузового потока, а вместе с этим в систему вошёл и модуль определения типа транспортного средства без внешнего сканера — только по видеоаналитике. В буклете 2022 года перечисляются варианты кузова, которые такой классификатор различает: фуры, автобусы, лёгкий коммерческий транспорт, легковые машины, внедорожники, мототехника. В более ранней публикации 2018 года компания писала о распознавании 392 типов номерных знаков 58 стран, а в буклете 2022 года уже заявлены 944 типа знаков 94 стран. Даже если отнестись к этим цифрам как к самоописанию производителя, по ним хорошо видно направление развития: от локальной российской системы фиксации — к более универсальному узлу идентификации и транзитного учёта.



«АвтоУраган-МС» — это уже не стационарная стойка, а прибор на крыше патрульного автомобиля. По описанию Autonews, такой комплекс получил гибридный функционал: обзор на 360 градусов, до шести камер, центральный доплеровский радар, работа и в движении, и на стоянке, передача данных в ЦАФАП или ЦОДД, хранение информации не менее

30 суток. Мобильная версия может фиксировать парковку, скорость, запрещённые манёвры и одновременно работать как инструмент розыска по базам.

Полной публичной схемы архитектур компания не раскрывает. Но по открытым публикациям видно, что к концу 2010-х дорожная отрасль в России в целом стала переходить от чистого OCR и логики набора жестких правил распознавания к нейросетевой аналитике типа транспортного средства, марки и модели. В заметке «Коммерсанта» 2018 года прямо говорится о внедрении нейросети для определения типа, марки и модели автомобиля по силуэту, логотипам и фонарям; там же представитель «Технологий Распознавания» Сергей Кусов отмечал, что в задачах контроля грузовых автомобилей и розыска при большом числе однотипных машин нейросеть «незаменима». Это не доказывает, что весь стек «АвтоУрагана» был полностью переведён на YOLO или SSD, но очень хорошо показывает направление: классическая система фиксации стала обрастать слоями CNN-аналитики там, где ручных признаков уже не хватало. В интервью Autonews 2020 года Кусов довольно откровенно описывает пределы автоматического распознавания грязных номеров. По его словам, система может по нескольким кадрам и косвенным признакам «достаивать» номер, но если нет явного доказательного кадра, такой косвенно дорисованный материал нельзя использовать для вынесения штрафа и он уходит в брак, хотя остаётся полезным для розыска. Это очень показательно для всей отрасли: между «понять, какая машина проехала» и «получить юридически достаточный материал» лежит большая разница. Для практического машинного зрения это один из самых полезных уроков, которые вообще можно вынести из истории дорожных комплексов.

«АвтоУраган» интересен как российский пример того, как прикладное машинное зрение взрослеет. Сначала оно решает одну задачу — читаем номер. Потом к этому добавляется метрология — скорость, время, синхронизация. Затем логика событий — перекрёсток, стоп-линия, средняя скорость, железнодорожный переезд. Потом слои аналитики — тип кузова, розыск, маршрут, работа с базами. И уже на следующем витке вся эта система начинает впитывать нейросетевые компоненты там, где нужно распознавать более сложные визуальные свойства машины и сцены. Такой путь очень типичен: не одна «революционная» модель, а постепенное наращивание функций вокруг надёжного ядра распознавания и доказательной фиксации. Именно по такой логике и траектории развитие движется система распознавания и аналитики бильярдных матчей и тренировок PerformStars.com, которую я имя команда развивает с 2020го года.

Вечная дилемма машинного зрения: «быстрый хак» против «революционной модели», где соблазн быстрого результата часто маскирует хрупкость решений. Хак выглядит заманчиво: всегда можно подогнать параметры под конкретные условия, добиться результата и даже убедить себя в его универсальности. Но стоит лишь слегка изменить освещение, угол камеры или фон — и детектор мгновенно слепнет, теряя всякую практическую ценность. Дэвид Марр предупреждал об этой ловушке в своей книге: пока мы не формализуем четко цель, внутреннее представление данных и механизм их обработки, мы неизбежно будем путать удачную эвристику с настоящим объяснением, рискуя застрять в цикле локальных заплаток. Цветовые «костыли» и прочие трюки с порогами плохо переносятся между доменами; истинная надежность рождается из обучаемых признаков, инвариантных к вариациям, и строгих тестов на переносимость, а не из нагромождения переменных, которые ломаются при первом же сдвиге контекста.

Эта дилемма, в сущности, отражает предрасположенность человеческого разума к универсальным решениям, корнящуюся в эволюционной истории: наш мозг, сформированный десятками тысяч лет адаптации к изменчивой среде, эволюционировал в сторону стратегий, которые минимизируют зависимость от локальных условий, предпочитая абстрактные модели, способные переноситься на новые сценарии. Почему так? Потому что в мире, где завтрашний день может принести засуху вместо изобилия или хищника вместо добычи, универсальность — это выживание; она позволяет строить ментальные карты, применимые шире, чем конкретный случай, повышая шансы на успех в непредвиденном. Это стремление проявляется не только в технологиях, но и в других сферах: в науке, где Ньютон стремился к единой гравитационной теории, объединяющей падение яблока и орбиту планет, вместо разрозненных описаний, или Эйнштейн — к общей теории относительности, интегрирующей пространство-время в универсальный каркас; даже в повседневной этике, где кантианский императив — «поступай так, чтобы максима твоей воли могла стать всеобщим законом» — возводит мораль в ранг универсального принципа, применимого за пределами личных обстоятельств. Таким образом, наше тяготение к автоматизации в машинном зрении — не просто инженерный выбор, а эхо этой фундаментальной человеческой склонности, побуждающей нас искать надежные мосты через хаос вариаций.

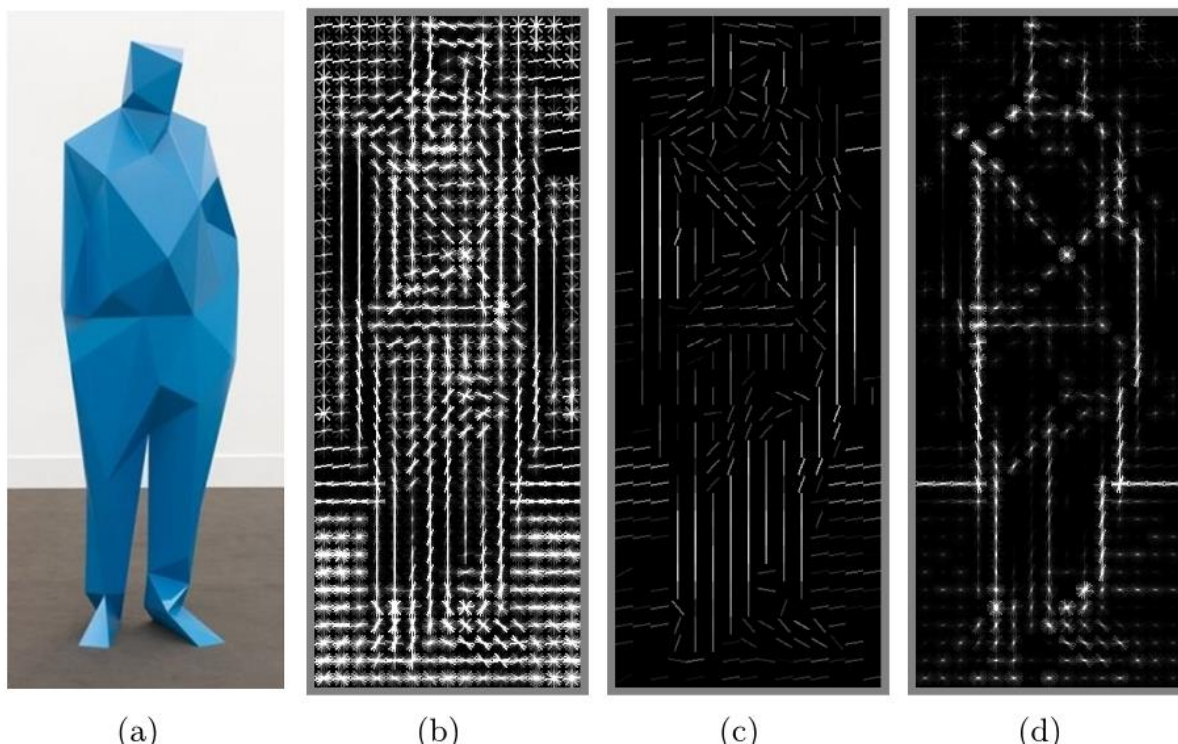
Начиная с бурного расцвета машинного зрения в 2010-х годах, когда глубокие нейронные сети начали перестраивать ландшафт компьютерного зрения, и до сих пор еще не появился **Общий искусственный интеллект** (ОИИ, **AGI** от англ. Artificial General Intelligence) — универсальный ИИ, который обладает самосознанием и способен охватить все

многообразие конкретных, узкоприкладных задач. Такой всеобъемлющей модели пока не существует, несмотря на прогресс в трансформерах и мультимодальных системах. Это фундаментальное ограничение побуждает нас к поиску локальных решений для локальных вызовов, которые ставит перед нами сама жизнь: детектор для распознавания лиц в толпе не подойдет для анализа траекторий шаров в бильярдном спорте, а сегментатор для медицинских снимков окажется бесполезным в автономном вождении. Дело здесь не в тривиальных вариациях переменных или изображений — освещение, шум или разрешение можно компенсировать постобработкой, — а в более высоком уровне абстракции, где алгоритм фундаментально не постигает человеческого целеполагания, той тонкой иерархии приоритетов, которая определяет, что именно мы ищем в визуальном потоке: безопасность ли, эстетику, эффективность или скрытый нарратив. Без понимания этой семантической глубины, где цель не сводится к пикселям, а рождается из контекста и намерения создателя, универсальность остается иллюзией; вот почему для разных классов задач — от реального времени в робототехнике до архивного анализа в историографии — мы неизбежно прибегаем к специализированным алгоритмам, каждый из которых, словно инструмент в мастерской, оттачивается под уникальный угол зрения, приближая нас к истине через фрагментарное, но точное постижение мира.

Вопрос достижим ли Общий Искусственный Интеллект, сейчас относится не к технической области, а к проблемам философии. Я уже упоминал, что философия и футурология входят в сферу моих активных интересов. Я основал движение «кремниевое братство» как философский дискуссионный клуб для живого обмена мнениями по вопросам технологического развития общества и продвижения лучших технических практик на благо мира и свободы. В отличие от искусственного интеллекта, сейчас мы ещё не знаем, какая материальная структура, лежит в основе нашего сознания и интеллекта на фундаментальном уровне. Поэтому, в обозримом будущем, Общий Искусственный Интеллект недостижим, а то что есть, — это иммитация и формализация отдельных логических способностей, к которым, со временем, вероятно добавятся ещё механические. Открытия на этом фундаментальном уровне стоит ожидать через прорывное развитие квантовой физики, математики, и во всех остальных, связанных с ними областях. Условия для кардинальных изменений в них должны накопиться вместе с общими социальными изменениями. Путь к этим изменениям — сотрудничество и обмен мнениями.

HOG

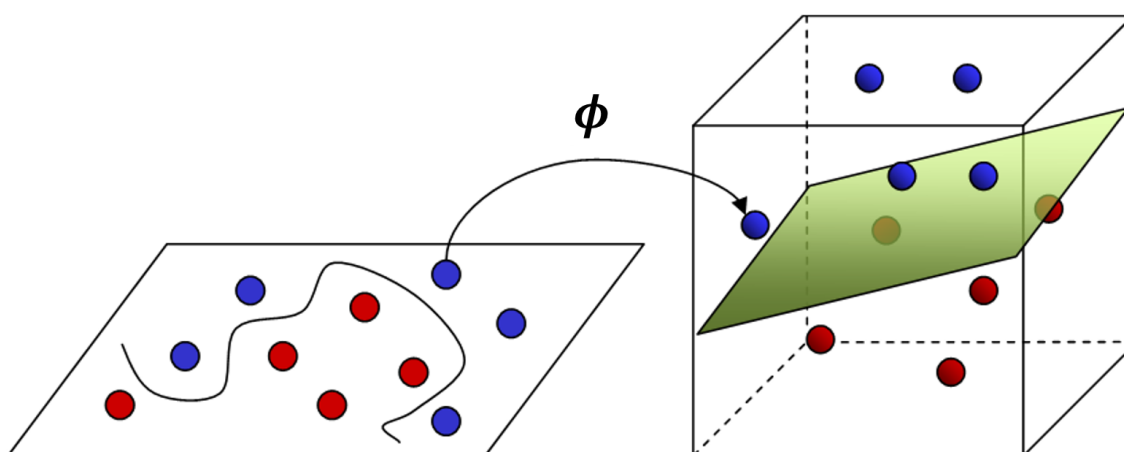
В 2005 Навнит Далал и Билл Триггс решали прикладную задачу — обнаружение людей на уличных сценах для видеонаблюдения и автомобильной безопасности. Пешеход — удобная цель: у него довольно стабильный силуэт (в полный рост, голова-плечи-туловище-ноги), но фон и освещение меняются постоянно. Предыдущие «общие» признаки (сырой PCA, простые прямоугольные контрасты в духе Виолы–Джонса) либо были слишком чувствительны к свету и позе, либо теряли форму. Нужен был описатель, который «цепляется» именно за очертания и при этом мало зависел от освещения. Так и родилась идея HOG.



HOG (Histogram of Oriented Gradients — гистограммы ориентированных градиентов) — это способ описать локальную форму как распределение направлений контуров. По сути, окно изображения превращается в «отпечаток» его краев. Сначала берут градиенты яркости в каждом пикселе — направление показывает ориентацию местной кромки, длина градиента — ее силу. Затем окно (для пешехода обычно 64×128 пикселей) делят на мелкие ячейки вроде 8×8 . В каждой ячейке накапливают гистограмму направлений. Чтобы приглушить влияние тени/контраста, соседние группы ячеек (блоки, например 2×2 ячейки) нормируют по контрасту и двигают их с перекрытием — так один и тот же край «виден» в нескольких блоках и становится устойчивее к неравномерному освещению. Наконец, все нормированные гистограммы склеивают в один вектор признаков: для классического окна это получается несколько тысяч чисел (порядка 3780).

Дальше — чистая математика: этот вектор подают в линейный SVM, обученный отличать «пешеход» от «не пешеход». Чтобы найти людей на всей картинке и при разных размерах, то же окно с теми же признаками сканируют по изображению на разных масштабах (пирамиде). Именно из-за устойчивости к освещению и фону, а также благодаря плотному покрытию окна HOG оказался особенно силен именно на пешеходах: их силуэт дает характерный «рисунок» направлений, который SVM ловит очень надежно.

Ранее я писал про Вапника–Червоненкиса и появление опорных векторов в 1990-е. Вот здесь, в связке HOG + SVM, эта теория применима. SVM (Support Vector Machine) ищет гиперплоскость с максимальным зазором между классами. Линейный SVM подбирает веса и смещение, чтобы оценка была положительной для «пешехода» и отрицательной для фона, причем с запасом. То есть балансируется ширина смещения и штраф за ошибки. В итоге на решение влияют лишь точки у границ контуров — опорные векторы; именно они «выкладывают» разделяющую плоскость. Если включить воображение, то можно представить работу SVM как изображено ниже, где цветами отражены разные признаки анализируемых объектов. Задача провести равноудаленную плоскость сквозь весь набор признаков, зная которую, можно пространственно классифицировать новые объекты, как относящиеся к соответствующей группе.



Почему это так хорошо работало именно с HOG? Во-первых, HOG дает сильно информативный, но все еще линейно разделимый вектор признаков: нормировка по блокам сглаживает освещенность, а гистограммы направлений фиксируют силуэт. Во-вторых, линейная модель означает, что проверка окна — это один скалярный продукт, что критично при сканировании пирамиды масштабов. Никаких ядер (kernel) здесь не требуется: «нелинейности» уже запакованы в сам признак

HOG; ядровой SVM давал бы лишь небольшой выигрыш в качестве ценной резкого падения скорости — неприемлемо для детекции.

Тренируется все относительно просто: берем много положительных окон с людьми и еще больше отрицательных со «всеми остальным». После первой подгонки прогоняем текущий детектор по большим фоновым кадрам, подбираем «сбивающие» ложные срабатывания и добавляем их в обучение. Это сдвигает границу туда, где она действительно нужна, и резко уменьшает шум в реальном сканировании. Линейный SVM не «пытается понять человека» — он математически проводит устойчивую границу там, где HOG-векторы «пешеходов» статистически отделимы от фона. Поэтому он так хорошо переносится между камерами и сценами: пока распределения признаков схожи, ширина зазора дает обобщение, а не подгонку под один датасет.

Мой первый практический опыт применения SVM — классификация номеров шаров по цвету. Я строил для каждого найденного шара простой вектор признаков: средний RGB внутри рамки детекции, нормированный по экспозиции и балансу белого через эталонные патчи — белый шар и участок сукна. Датасет собирал из реальных партий: на каждом кадре у меня уже были облака точек и номера шаров, поэтому я мог уверенно отправлять цветовой вектор на обучение соответствующих меток. SVM обучал в мультиклассовой постановке «один против всех». Это означает что на выходе у классификатора имеется несколько классов, в отличие от бинарной классификации, когда алгоритм выбирает только один из двух вариантов, обычно это ноль и единица. Истина и ложь. Один против всех означает, что из множества классов можно выбрать только один, то есть общая сумма вероятностей по всем классам даёт единицу. Есть ещё постановка «многие ко многим», где значение может одновременно относиться к нескольким классам. В таком случае максимальное значение суммы всех классов может быть числом классов, при условии, что объект попадет по всем классам сразу с вероятностью единица на каждый. Постановка «один против всех» немного облегчает выбор итогового результата и работу с ним. Но если важнее точность, то «многие против многих», могут дать чуть больше данных для более сложной пост обработки. Искусство машинного зрения заключается в том, чтобы правильно сориентироваться и смоделировать классификатор, сообразный реальной ситуации его применения.

Что касается моего классификатора, на валидации в рамках одного матча результат был умеренно хорошим и предсказуемым: белый шар выделяется уверенно, контрастные цвета вроде синего держатся стабильно, а соседние по тону классы начинают путаться при теплом свете, бликах и легком размытии. Особенно похожими были красный, оранжевый и розовый шары. В итоге я столкнулся с проблемой широкой

«палитры пирамид»: у разных наборов шаров отличаются пигменты, сукно дает цветовой сдвиг, камеры и кодеки по-разному ведут темные участки. Универсальный межклубный классификатор по одному только среднему RGB не получается: распределения классов перекрываются сильнее, чем позволяет разделить статическая гиперплоскость. Зато внутри одного матча или одной площадки идея жизнеспособна: быстрая краткая подстройка SVM по нескольким размеченным кадрам дает аккуратный локальный классификатор, который дешев в эксплуатации и прозрачен для отладки.

Дальнейшие действия пошли по трем направлениям. Во-первых, это добавление к среднему цвету еще пары простых статистик — дисперсии и доли «бликов» в маске шара — это стабилизирует схему на глянце и при высоком ISO. Во-вторых, это применение связки цвета с эмбедингом: компактный эмбединг из легкой сети, обученной на кропах шаров с аугментациями освещения, хорошо дополняет SVM как линейный решатель в пространстве более информативных признаков. Такой гибрид обычно снижает зависимость от конкретной камеры. В-третьих, добавление адаптации под каждое видео: несколько эталонных кадров для построения локального SVM — и весь матч я могу отличать идентификаторы шаров.

NMS подавление немаксимумов

В работе многие детекторы объектов возвращают сырые оценки координат пограничных рамок, вынуждая использовать пороговый выбор из множества кандидатов и последующее объединение перекрывающихся окон NMS — *Non-Maximum Suppression*, по-русски чаще говорят «подавление немаксимумов». В компьютерном зрении этим словом называют важную функцию финальной обработки результата. Например, в детекции объектов (то, о чем мы говорили с HOG+SVM и дальше) разные модели выдают много перекрывающихся прямоугольников-кандидатов с оценками уверенности. NMS оставляет один «главный» бокс и убирает дубликаты вокруг него. Логика простая: сортируем окна по уверенности, берем лучшее, все остальные с сильным перекрытием с ним (по метрике IoU как отношение площади пересечения к площади объединения) и выше порога, скажем 0,5, — подавляем. Повторяем, пока кандидаты не кончатся. Так снимаются «пучки» окон вокруг одного и того же пешехода/лица. Порог площади пересечения/объединения — гиперпараметр; часто делают NMS отдельно по классам; перед NMS отбрасывают слабые окна (pre-NMS top-k) ради скорости. Из вариантов: **Soft-NMS** не выкидывает, а плавно понижает пороги перекрывающимся боксам; **Weighted Box Fusion** усредняет координаты нескольких сильных окон вместо выбора одного; в трансформер-детекторах вроде **DETR** пытаются вообще обойтись без NMS, обучая модель выдавать набор неперекрывающихся предсказаний.



В выделении границ (детектор Кэнни) тоже используется NMS — это локальное «истончение» края: вдоль направления градиента оставляем только пиксель-максимум и подавляем соседей, чтобы контур стал одинарной тонкой линией. Идея та же — оставить максимум, убрать «почти-максимумы» вокруг, но применяется к пикселям, а не к боксам.

В качестве альтернативы классическим каскадам Viola-Jones я провел несколько экспериментов с подходом на основе HOG и SVM для обнаружения шаров на кадрах. Цепочка алгоритмов здесь действительно получается проще и интуитивно понятнее: всего несколько шагов

от извлечения дескрипторов до классификации. Однако общий результат оказался менее надежным — модель трудно адаптировать к универсальным требованиям, поскольку она требует тщательной предварительной подготовки кадра, включая нормализацию перспективы и устранение шумов, чтобы дескрипторы не искажались. В идеальных условиях, на строго откалиброванном ракурсе сверху вниз с фиксированным освещением, точность могла достигать и 90 процентов, но при малейшем отклонении — смене угла камеры или вариациях фона — производительность резко падала, делая подход непригодным для универсального приложения, способного прилипать на любой ракурс из случайных трансляций.

SURF

В то время, когда в 2005 году HOG ввел градиентно-ориентированные дескрипторы для устойчивого описания форм объектов, параллельно эволюционировали методы извлечения ключевых точек, ориентированные на инвариантность к масштабам и ротациям, — здесь SIFT, появившийся в конце 1990-х, уже зарекомендовал себя как эталон благодаря своей способности выделять устойчивые ключевые точки и генерировать детализированные «отпечатки» локальных окрестностей на основе разности гауссианов и 128-мерных дескрипторов, но в чистом виде он оставался вычислительно затратным, особенно для задач реального времени. Чтобы преодолеть эту тяжесть, в 2006 году Герберт Бай, Тинне Тейтеларс и Люк ван Гул изобрели SURF — Speeded-Up Robust Features, — элегантный компромисс — набор аппроксимаций, опирающихся на те же интегральные изображения, что и Viola-Jones, для ускорения вычислений. Вместо точных гауссовских фильтров SIFT, SURF использует простые коробчатые фильтры для приближения вторых производных — детектор основан на матрице Гессе для быстрого поиска ключевых точек, а дескриптор строится из сумм градиентов по горизонтали и вертикали в 4x4 сетке, сжатой до 64 или 128 измерений с учетом знака для повышения дискриминативности; эти упрощения снижали время обработки на порядок — до 3–5 раз быстрее SIFT — при потере качества всего в 5–10 процентов по метрикам сопоставления, делая метод подходящим для эпохи, когда камеры в смартфонах и веб-камерах множились экспоненциально, а приложения от склейки панорам в Photoshop до трекинга объектов в видео требовали приличной скорости на скромных процессорах или даже мобильных чипах без специализированного железа. SURF таким образом воплощал идеальный компромисс «почти SIFT, но быстрее и легче встраиваемый», открывая двери для коммерческих продуктов вроде систем дополненной реальности или робототехники, где задержка в миллисекунды имела значение. Интересный исторический штрих: как и SIFT, SURF долгое время скрывался за патентами — Левенский католический университет в Бельгии был его совладельцем до 2021 года, — из-за чего в OpenCV оба алгоритма помещали в проприетарный модуль xfeatures2d, а истечение патента SIFT в 2020 не только вернуло его в основную ветку, но и спровоцировало всплеск интереса open-source сообщества к SURF, который последовал годом позже.

CUDA



Главный сдвиг 2000-х в машинном зрении произошел благодаря фундаментальной трансформации инфраструктуры вычислений и обработке данных, которая радикально расширила горизонты возможного. В июне 2007 года NVIDIA официально выпустила CUDA — Compute Unified Device Architecture, — это был прорывной фреймворк, открывающий разработчикам доступ к параллельным вычислительным мощностям графических процессоров (GPU) для задач общего назначения, выходящих за рамки чистой графики. Сам термин GPU придумали маркетологи NVIDIA, по созвучности с CPU, только с буквой G - графика. До этого GPU использовались преимущественно для рендеринга игр и визуализации, но CUDA позволила перепрограммировать тысячи ядер на чипе как единую вычислительную платформу, ориентированную на расширенный C/C++, с многопоточным режимом выполнения задач для максимальной эффективности. Научные группы в области компьютерного зрения мгновенно осознали потенциал: ключевые операции, такие как свертки (convolutions) для извлечения признаков, применение фильтров (например, собелевских или гауссовских для сглаживания), построение пирамид изображений (многомасштабных представлений для детекции объектов разного размера) и даже матричные умножения в SVM, идеально ложатся на параллельную архитектуру GPU, где тысячи простых вычислений выполняются синхронно. В результате, практика ускорения кода в 10–100 раз по сравнению с процессором, перестала быть экзотикой и вошла в повседневную практику: простая реализация свертки на

CUDA могла сократить время обработки видео с часов до минут, решая задачи, ранее считавшиеся непосильными. Уже к 2008 году появились бета-версии CUDA для macOS, расширяя экосистему за пределы Windows и Linux, а за следующие несколько лет GPU эволюционировали от нишевого инструмента к индустриальному стандарту для ресурсоемких расчетов — особенно для обучения нейронных сетей, где итеративная оптимизация весов через градиентный спуск требует миллиардов операций за каждый цикл обучения. Именно эти чипы сделали глубокие нейронные сети по-настоящему глубокими: без их параллелизма слои в десятки и сотни нейронов оставались бы недоступными из-за взрывного роста вычислений, а прорывы вроде AlexNet в 2012 году на ImageNet стали возможны только благодаря CUDA-оптимизированным фреймворкам вроде cuDNN. На фоне типичных тактовых частот CPU в сотни мегагерц и первых многоядерных процессоров вроде Intel Core 2 Duo с 2–4 ядрами, где последовательные операции доминировали, GPU с тысячами ядер по 1–2 ГГц обеспечили качественный скачок — не просто ускорение, а переход за грани возможного.

В качестве иллюстрации приведу пример из своей практики, как в конце 2023 года, в ходе целенаправленных экспериментов по оптимизации серверной инфраструктуры, мне удалось интегрировать два графических процессора NVIDIA GTX 1080 Ti в вычислительный кластер и скомпилировать специализированную версию фреймворка Caffe под операционной системой Ubuntu 20.04, с полной поддержкой CUDA 11.2 для ускорения вычислений на GPU. Эта конфигурация позволила эффективно обучать и разворачивать нативные модели на базе VGGNet — архитектуры, о нюансах которой я расскажу подробнее в следующих разделах, — без необходимости перехода на более современные платформы. Переход к GPU-ускорению радикально преобразил процесс детекции бильярдных шаров на кадрах видео: среднее время анализа одного изображения сократилось в 30 раз — с 1,1 секунды на CPU до всего 50 миллисекунд на GPU, — что стало возможным благодаря параллелизации свертки и матричных операций, где тысячи ядер 1080 Ti обрабатывали карты признаков одновременно, минимизируя латентность без потери точности. Изначально проект стартовал на процессоре Intel Xeon 2978 с частотой 2500 МГц.



PASCAL VOC

Во второй половине 2000-х годов развитие компьютерного зрения вступило в эпоху «больших общих задач», когда научное сообщество осознало острую необходимость в стандартизации подходов и объективных критериях оценки. До этого прогресс часто измеряли на разрозненных, несопоставимых датасетах — небольших лабораторных коллекциях изображений, собранных в конкретных условиях, таких как коридоры университета или студийные сетки, — что приводило к ситуациям, когда один алгоритм блистал в узком тесте, но проваливался в реальности, а сравнения между группами превращались в споры о «своих» против «чужих» данных. Чтобы преодолеть этот хаос, в 2005 году группа европейских исследователей под эгидой проекта PASCAL (Pattern Analysis, Statistical Modelling and Computational Learning) запустила ежегодный конкурс PASCAL Visual Object Classes (VOC), который стал настоящим поворотным моментом. Это был не просто набор тестов, а полноценная платформа с едиными правилами: открытыми, публично доступными датасетами изображений, четко определенными метриками успеха — такими как средняя точность (mean Average Precision, mAP), учитывающая как полноту обнаружения объектов, так и минимальное пересечение с «истиной в последней инстанции» (IoU), — и строгим разделением на тренировочные, валидационные и тестовые подмножества, чтобы избежать подгонки под конкретные примеры.

Первые выпуски VOC-2005 были скромными по масштабам — всего четыре класса объектов для детекции (велосипед, человек, машина, автобус) и базовые задачи вроде классификации и сегментации, — но уже к 2007 году конкурс эволюционировал в легендарный стандарт с двадцаткой категорий, отражающих повседневную реальность: от людей и транспортных средств (машины, самолеты, поезда) до домашних животных (кошки, собаки, лошади), предметов интерьера (стулья, бутылки, обеденные столы) и даже уличных элементов (машины, автобусы). Впервые исследователи соревновались на равных условиях — на одном общем поле, с аннотированными изображениями из реальных источников, включая фото с камер и интернета, — что позволило объективно сравнивать алгоритмы не по «коридорным» базам данных, а по способности справляться с разнообразием поз, масштабов, окклюзий и фонов. Для индустрии это означало появление надежного бенчмарка, напрямую коррелирующего с практическими сценариями: системы видеонаблюдения, автономные автомобили или поисковые движки теперь могли опираться на метрики, предсказывающие реальную производительность. Такой подход радикально ускорил естественный отбор идей — полезные инновации, устойчивые к вариациям, быстро поднимались

в рейтингах и переходили в продукты, в то время как хрупкие, домен-специфичные трюки отсеивались без жалости, очищая поле для настоящих прорывов.



Эта стандартизация в компьютерном зрении, подобная PASCAL VOC, где разрозненные эксперименты уступили место общему бенчмарку с едиными классами объектов, метриками и данными, неизбежно вызывает параллели с механизмами сортировки и ранжирования в человеческом сообществе, особенно в спорте, где эволюционная потребность в иерархии находит свое наиболее чистое выражение. В спорте этот процесс обретает кристальную ясность: турниры и лиги, словно датасеты VOC, вводят строгие правила — от измерения времени в беге до подсчета очков в командных играх, — чтобы превратить хаос индивидуальных усилий в объективную градацию, где каждый спортсмен или команда оценивается по единой шкале, исключая субъективные «коридорные» тесты вроде неформальных спаррингов. Именно такая унификация метрик — как mAP в зрении, так и статистика владения мячом в футболе — позволяет не только выявлять таланты, но и ускорять эволюцию дисциплины: подобно тому, как на примере сравнения результатов применения нейросетевых алгоритмов на одинаковом датасете PASCAL VOC, слабые алгоритмы уступали место более устойчивым.

Применяя эту аналогию к моему проекту по бильярдной аналитике, который автоматизирует анализ ударов через траектории, силы и точность на основе стандартизированных критериев, внедрение его в реальный спорт не только целесообразно, но и способно привести к локальной трансформации: росту популярности через прозрачность и зрелищность (как в шахматах с компьютерным анализом), повышению конкурентности за счет персонализированных тренировок и, возможно, переосмыслению эталонов успеха — где не только удача в партии, но и долгосрочная статистика станет определять новую бильярдную элиту. Я

уверен, что благодаря моему проекту, мир бильярда узнает много новых имен, которые всплывут из провинциальных районов. Ими станут игроки, которые ранее не имели возможности быть публично квалифицированными, но не уступают по уровню мировым лидерам.

Когда в конце 2022 я начал экспериментировать с нейросетевыми детекторами шаров, самым доступным и предсказуемым вариантом оказался готовый извлекатель признаков — нейросеть VGGNet, обученная на датасете PASCAL VOC 2007, где перечисленные выше двадцать бытовых классов уже умеют различать по внутренним признакам базовые образы; к ним я добавил свой двадцать первый класс — бильярдные шары. Нейросеть уже сформировала веса, которые умеют видеть кромки, текстуры и простые композиции сцены; остается доучить имеющиеся слои под новую задачу. Чтобы все заработало, я привел свою разметку к нотации VOC: изображения в JPEG, аннотации в XML с объектами, именем класса и прямоугольниками `bndbox` в пиксельных координатах, плюс аккуратная укладка в привычную структуру каталогов с индексами в `ImageSets/Main`. Такой формат дружит практически со всеми открытыми пайплайнами и позволяет сразу включить стандартную оценку качества по метрике AP при пороге перекрытия 0,5 — тот самый «PASCAL-критерий», от которого пошли многие сегодняшние метрики.

В дальнейшем, в процессе научного поиска и сравнения подходов, я перешел на MS COCO — с богаче устроенной метрикой AP на диапазоне порогов перекрытия. Но именно PASCAL VOC дал мне правильный старт: стабильный формат аннотаций, воспроизводимую метрику, готовые веса VGG и большой набор отлаженных инструментов. Этот этап позволил быстро «поднять» рабочий детектор шаров, навести порядок в датасете и только затем двигаться к более требовательным и современным корпусам, о которых я расскажу далее.

ImageNet



Почти синхронно с запуском PASCAL VOC, в 2009 году исследовательница Фэй-Фэй Ли вместе с коллегами из Принстонского университета и Стэнфорда представила ImageNet — масштабную базу данных изображений, тесно интегрированную с иерархической структурой WordNet, семантической сетью английских слов, которая организует понятия в виде дерева: от общих категорий вроде «животное» до конкретных, таких как «сибирский хаски», с учетом синонимов и родственных связей. Это позволяло не просто собирать фото, а строить когерентную, иерархическую коллекцию, где каждый класс визуальных объектов подкреплен лингвистическим контекстом, облегчая задачи вроде семантической сегментации или иерархической классификации. В первом докладе на конференции CVPR 2009 года они продемонстрировали уже впечатляющий прогресс: 3,2 миллиона размеченных изображений, распределенных по 5247 синсетам (группам синонимичных понятий из WordNet), с фокусом на 12 поддеревьях иерархии, таких как «существа» или «предметы»; при этом каждое изображение аннотировано не только меткой класса, но и рамкой для локализации объекта. Особый акцент в докладе сделали на процессе разметки: чтобы достигнуть требуемого объема данных, они применили краудсорсинговую платформу Amazon Mechanical Turk, где тысячи фрилансеров-аннотаторов выполняли микро-задачи — от подтверждения релевантности фото до рисования рамок вокруг объектов, — с многоэтапной верификацией (например, кросс-проверкой несколькими исполнителями и алгоритмическим контролем несоответствий), что сделало процесс масштабируемым, и позволило избежать болезненных ошибок в структуре обучающих данных.

Масштаб поражал воображение даже тогда: изначальная цель — расширить базу до 14 миллионов изображений по 21 841 синсету, с минимум 500–1000 примерами на класс, чтобы обеспечить статистическую мощь для обучения сложных моделей; это было не просто хранилище,

а фундамент для будущих исследований, где разнообразие источников (от Flickr до поисковиков) гарантировало репрезентативность реального мира. В тот момент проект казался прежде всего педантичной, почти библиотечной работой — методичным каталогизированием визуального мира во имя науки, без немедленного коммерческого результата. Однако именно из этой инфраструктуры через пару лет, в 2010 году, вырос ILSVRC — ImageNet Large Scale Visual Recognition Challenge, ежегодный международный конкурс по крупномасштабной визуальной классификации и детекции, где участники соревновались на подмножестве в 1000 классов с миллионами изображений, используя строгие метрики вроде top-1 и top-5 accuracy; этот вызов стал очередным катализатором, открыв эпоху глубокого обучения в компьютерном зрении, поскольку огромные данные наконец-то позволили нейронным сетям «выучить» сложные паттерны без ручной инженерии признаков. В ретроспективе это выглядит как классический паттерн истории технологии машинного зрения: сначала появляется хороший датасет для фундамента, затем соревновательный конкурс для проверки идей и алгоритмов на базе общей сравнительной платформы, а потом — настоящая лавина инновационных архитектур, от AlexNet в 2012 году до современных трансформеров.

ILSVRC

Десятилетие 2010-х годов в компьютерном зрении открылось не столько революционными алгоритмами, сколько фундаментальным актом консенсуса: сообщество, опираясь на растущую базу ImageNet, которая выше отмечена как ключевой инфраструктурный прорыв конца 2000-х, решило четко определить, как именно измерять прогресс, чтобы избежать разрозненных сравнений и субъективных оценок. Этот шаг материализовался в виде ежегодного соревнования ImageNet Large Scale Visual Recognition Challenge — ILSVRC, запущенного в 2010 году под эгидой ImageNet и конференций вроде CVPR, где участникам предлагалась единая, грандиозно амбициозная задача: классифицировать изображения по одной из тысячи разнообразных категорий — от повседневных объектов вроде «аккордеона» или «бензопилы» до сложных сцен с животными и транспортными средствами — на основе сотен тысяч реальных фотографий, собранных из интернета и аннотированных с помощью краудсорсинга. Идея «большой общей задачи» — единого полигона для всех — оказалась на удивление продуктивной, превосходя по влиянию десятки специализированных датасетов: крупные лаборатории вроде Microsoft Research или Google соревновались на равных с небольшими университетскими командами, а каждый выигранный процент точности превращался не в изолированную лабораторную цифру на слайде, а в общепризнанную веху, мотивирующую сотни последующих работ и публикаций. Новизна подхода проявилась и в растущем числе участников: если в 2010 году стартовало всего около 20 команд, то к 2012 их число превысило 100, включая неожиданных новичков из Азии и Европы, что ускорило обмен идеями через открытые репозитории кода на GitHub.

Эпоха открытых данных и научной воспроизводимости результатов утвердилась не в абстрактных манифестах или статьях, а именно на арене ILSVRC, где прозрачность правил выявила слабости традиционных методов — от ручной инженерии признаков до линейных классификаторов, — и заложила почву для неизбежного «землетрясения»: через пару лет, в 2012, победа AlexNet с глубокими сверточными сетями перевернет всю отрасль, сделав нейронные сети новым стандартом, и откроет путь к эре, где машинное зрение интегрируется в повседневные приложения от смартфонов до автономных автомобилей.

AlexNet

В 2012 году в мире компьютерного зрения разразился настоящий переполох, и его катализатором стала AlexNet — прорывная модель, которая перевернула представления о возможностях сверточных нейронных сетей (CNN, convolutional neural networks). Хотя сама концепция CNN была известна с 1980-х годов благодаря работам Яна Лекуна, применение их на таких масштабах — с миллионами параметров и крупным датасетом ImageNet — оставалось террой инкогнито, ограниченной вычислительными барьерами. Команда из Университета Торонто — Алексей Крижевский (родился в 1986 году в СССР, в начале 90-х его семья эмигрировала в Канаду), Илья Суцкевер (родился в 1986 году в СССР, в Горьком — ныне Нижний Новгород, Россия, и тоже эмигрировал в Канаду с родителями в детстве) и их наставник Джеффри Хинтон, лауреат Тьюринговской премии, — представили восьмислойную архитектуру, обученную на двух относительно доступных видеокартах NVIDIA GTX 580, каждая с тремя гигабайтами видеопамяти.



Это был скромный по сегодняшним меркам двухвидеокартный компьютер в университетской лаборатории. Процесс тренировки растянулся на пять-шесть дней с использованием параллельного вычисления. Сеть насчитывала около шестидесяти миллионов параметров, что требовало чрезвычайных мер против переобучения: случайное «выключение» нейронов во время обучения (слой dropout, который с вероятностью 50% обнуляет веса нейронов, заставляя модель полагаться на разнообразные пути), агрессивная аугментация данных — случайные обрезы, горизонтальные отражения и сдвиги изображений для имитации

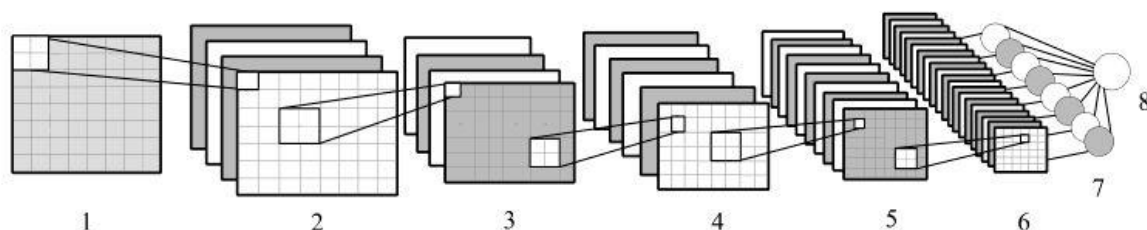
вариаций в реальном мире, — а также нелинейные активации ReLU (Rectified Linear Unit), где функция просто обнуляет отрицательные значения весов, предотвращая «затухание градиентов» при обучении и ускоряя сходимость втрое по сравнению с традиционными подходами. Результат на конкурсе ILSVRC 2012 ошеломил даже самых закоренелых скептиков: ошибка в топ-5 классификации составила всего 15,3 процента против 26,2 процента у ближайших конкурентов, использующих комбинации ручных дескрипторов вроде SIFT и HOG с линейными классификаторами. В мгновение ока сообщество осознало: комбинация больших данных, GPU-ускорения и сквозного обучения — это наиболее надежный из известных инженерных путей к уровню точности, который ранее казался недостижимым на шумных, неконтролируемых изображениях из реальной жизни, где объекты скрыты, искажены или сливаются с фоном.

Ключевой инженерной изюминкой AlexNet стала адаптация к аппаратным ограничениям: модель просто не помещалась в память одной видеокарты, поэтому ее «разрезали» пополам — свертки распределялись по GPU, а промежуточные активации передавались между ними, — превратив узкое место железа в элегантное архитектурное решение, которое позже эволюционировало в полноценные фреймворки вроде TensorFlow и PyTorch.

Стоит отдельно отметить вклад Ольги Русаковской, российско-американского исследователя, соавтора ключевой статьи о ILSVRC 2014, где описаны результаты соревнования с участием более 50 команд. Она внесла вклад в организацию конкурса, фокусируясь на стандартизации метрик, таких как mean Average Precision (mAP). Хотя полноценных команд из России в топах ILSVRC не фиксировалось, отдельные исследователи из российских вузов (например, МФТИ) участвовали в связанных работах, адаптируя модели для задач вроде распознавания дорожных знаков на основе датасетов, вдохновленных ImageNet.

CNN

Тут важно провести разделительную линию в понимании классов задач, решаемых при помощи машинного зрения. Ранее я уже о них писал - детекция объектов, нахождение ключевых точек, описание изображений. AlexNet на базе датасета ImageNet решал задачу классификации единичных образов. Это определялось самой структурой датасета - есть изображение и его класс, на базе того, что на нем изображено. Изображение для компьютера — это просто таблица чисел: для каждого пикселя три значения яркости по каналам. Базовая задача — превратить эту таблицу в понятный ответ: что на кадре (классификация), где находятся объекты (детекция). Самая очевидная логика — «соединить» каждый пиксель со всеми нейронами и учить огромную сетку на миллиардах связей. На практике так не работает: слишком много параметров, слишком мало данных, слишком высокая цена обучения. Сверточная нейросеть предлагает более экономную логику: она смотрит на картинку локально, небольшими окошками, и использует одни и те же коэффициенты для всех мест изображения. За счет этого резко сокращается число настраиваемых параметров и появляется устойчивая «геометрия» обработки.



Базовый шаг — свертка. Берем маленькое окошко, скажем 3×3 пикселя, умножаем его значения на восемь коэффициентов «фильтра» (кружочки на схеме) и суммируем. То же самое делаем, сдвигая окошко на один пиксель вправо, вниз и так по всей картинке. В результате получается новая карта — она показывает, как сильно данный фильтр «узнает» свой шаблон в каждой точке кадра. Один фильтр будет откликаться на горизонтальные границы, другой — на наклонные, третий — на светлые кружки на темном фоне. Коэффициенты всех фильтров не задаются вручную: сеть сама подбирает их на примерах, чтобы ошибка на обучающем наборе уменьшалась. Количество фильтров выбирается произвольно, под конкретную задачу. Чем больше их количество, тем больше вариантов признаков получится ими описать, но тем больше будет вычислений.

Эта процедура повторяется слоями. Сначала сеть учится замечать простые контрасты и линии, затем — более сложные сочетания,

вроде дуг или пересечений, еще выше — характерные фрагменты объектов. С каждым слоем «поле зрения» увеличивается: итоговый признак в точке начинает зависеть не от 3×3 пикселей, а от достаточно большого фрагмента кадра. Параллельно сеть может постепенно уменьшать разрешение внутренних карт, чтобы охватывать контекст и экономить вычисления. Иногда это делают отдельным «сжатием» карт, иногда — шагом свертки больше единицы. На границах кадра вводят заполнение, чтобы не «терять» края при каждом проходе.

Цветовые каналы и внутренние карты обрабатываются согласованно: один и тот же фильтр смотрит сразу на все входные слои, но выдает один отклик. Несколько фильтров — несколько откликов. Так строится «пакет» карт признаков для следующего слоя. Важная особенность такого устройства — если объект на изображении сдвинут немного в сторону, отклик фильтра просто сдвинется вместе с ним. Для задач классификации этого часто достаточно, а для задач точного позиционирования (детекция, сегментация) добавляют пути, которые сохраняют более детальные карты и объединяют их с «крупным планом» на поздних этапах. Получается внутреннее представление сразу на нескольких масштабах: сеть знает и «что», и «где».

Обучение идет одинаково для всех слоев: сеть получает пример с правильным ответом, сравнивает свое предсказание с ним, вычисляет ошибку и изменяет коэффициенты фильтров так, чтобы в следующий раз ошибиться меньше. Никаких заранее заданных «краев» и «кружков» в код не зашито — понимание о них возникает как побочный продукт того, что наличие или отсутствие этих элементов помогает уменьшать ошибку при угадывании правильных ответов. Этим сверточная сетка отличается от метода ручных признаков: не надо описывать параметры для всех возможных образов, достаточно задать разумные ограничения на форму вычислений и позволить данным подстроиться под форму.

На практике этот принцип удобен тем, что он естественно масштабируется. Если увеличить размер входного кадра, свертки продолжают работать так же: фильтры переиспользуются по всей площади, вычислительная нагрузка растет линейно, а не квадратично по числу параметров. Это хорошо ложится на параллельные процессоры: одна и та же операция делается для множества окошек одновременно. За счет этого сверточные сети стали стандартом в распознавании сцен, в локализации объектов, в сегментации и в задачах типа «свободное пространство сверху».

Низовые слои быстро учатся подавлять шум и подчеркивать границы шаров; средние — реагировать на круглые контуры разных размеров и оттенков; более высокие — выделять комбинации «круг плюс

ть», «круг рядом с бортом», «форма лузы». Эта часть работы называется извлечением признаков. Далее под задачу можно обучить разные выходные слои — «головы», которые смогут решать разные задачи. Для этого данные для обучения должны быть в том же формате, что и ожидается получить на выходе. Если мы хотим делать классификацию изображений, то достаточно обучать нейросеть только итоговым классам этих изображений. Если хотим детектировать объекты, то в датасете добавляем координаты наших эталонных объектов. Соответственно, если задать «голову» для детекции образов, сеть начнет предсказывать координаты рамок, как в датасете; если задать «голову» только для классов, сеть будет предсказывать что нарисовано на картинке; если задать «голову» для масок, сеть будет закрашивать пиксели шара; если сложить несколько кадров, можно выучить и короткую динамику. Ниже мы подробнее поговорим о каждой из этих задач в контексте сверточных нейросетей.

Есть и ограничения, которые важно понимать. Сеть чувствительна к тому, как распределены данные: если освещение, цвета и ракурсы на обучении и в эксплуатации сильно различаются, то ее качество падает. Чем глубже сеть и чем больше разрешение, тем дороже обучение и инференс; баланс между точностью и скоростью приходится подбирать под устройство и сценарий.

CNN Классификация: когда всему изображению нужна одна или несколько меток

Классификация — самая понятная и, пожалуй, самая коварная задача машинного зрения. Мы просим модель посмотреть на изображение целиком и назвать, что на нем изображено. Иногда ответ должен быть единственным — «кошки», «пневмония», «дефект отсутствует». Иногда ответ предполагает выбор сразу нескольких классов: на снимке может быть и «пляж», и «закат», и «люди». В первом случае модель выбирает ровно одну категорию, во втором — отмечает сразу несколько. Это различие кажется технической деталью, но оно определяет и формат разметки, и то, как обучаем нашу модель, и даже то, как потом измеряем качество результата.

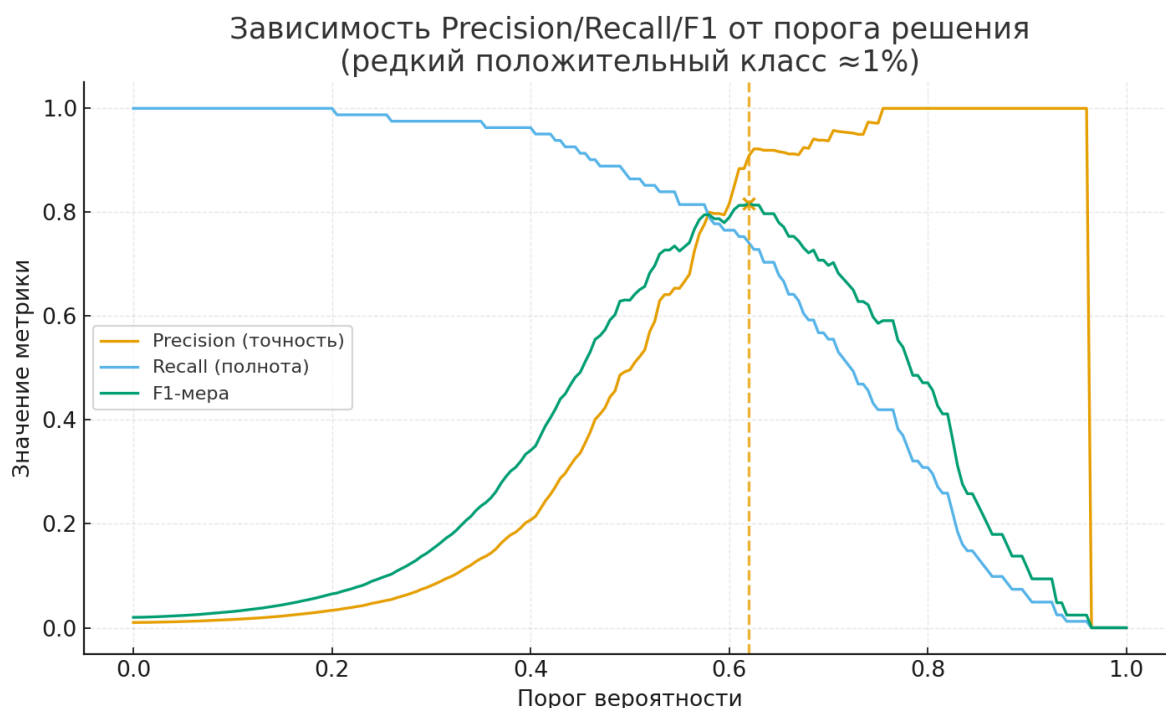
Разметка для классификации выглядит просто: у каждого изображения есть список меток. Для одноклассового варианта — одна метка, для многоклассового — набор, иногда длинный. Главное — заранее определиться с правилами. Что считать «кошкой»: только животное целиком или и крупный фрагмент мордочки? Считать ли «сумерки» разновидностью «ночи» или отдельным классом? Эти понятийные договоренности должны быть записаны и понятны разметчикам, иначе в данных появится шум, который влечет неустойчивость модели. Хорошая практика — показать аннотаторам десяток пограничных примеров с пояснениями «почему так» и «почему не так», а затем проверить, насколько они согласны между собой: если двое из трех систематически расходятся, значит, мы не договорились о критериях.

Вот конкретный пример из медицины: у нас есть тысячи рентгеновских снимков грудной клетки, и для каждого врач указал «пневмония» или «норма». Это классическая одноклассовая задача: модель должна вернуть вероятность каждого варианта, и в работе мы проводим границу — скажем, все, что выше определенного порога по «пневмонии», отправляется на верификацию старшему врачу. По похожему принципу работают классификаторы нежелательного контента: изображение помечается как допустимое или нет, иногда с более тонкой градацией, например нейтральное. В индустриальном контроле качества логика та же: деталь «годна» или «брак», а дальше уже технолог вручную решает, какую ветку процесса запустить.

Для многоклассовой постановки задачи приведу другие примеры. Представьте фотобанк, где одно фото может одновременно быть «природой», «морем» и «людьми». Модель не обязана выбирать одного «победителя» для каждой категории, она оценивает каждую метку отдельно и возвращает набор вероятностей для каждой фотографии. Здесь особенно важно думать не только о том, как обучить модель, но и о том, как вы будете использовать ее ответы: если редактор хочет видеть только

кадры с людьми на пляже, пороги для «людей» и «пляжа» придется подбирать отдельно, исходя из того, сколько пропусков и ложных срабатываний готовы терпеть.

Как понять, хорошо ли работает классификатор? Привычно и понятно ориентироваться на долю верных ответов, но для этого надо быть хорошо уверенным и понимать баланс своих данных. Если брак встречается в одном проценте случаев, система, которая всегда говорит «годно», покажет 99% точности и будет совершенно бесполезной, поскольку именно этот упущенный 1% пропуска это 100% ошибки. Или 0% реальной точности. Поэтому вместе с точностью смотрят на баланс точности и полноты, известную как F1-мера: она «наказывает» модель за то, что та либо пропускает нужные примеры, либо плодит ложные тревоги. На графике 1 пример, как меняются ключевые метрики в зависимости от порога решения; пунктир указывает порог с максимальной F1. Это и есть оценка точности результата. Когда классы редки, в качестве оценки удобны также кривые, показывающие, как меняется качество при сдвиге порога решения: по одной из них видно, насколько хорошо модель в принципе отделяет «свои» от «чужих», а по другой — как она ведет себя именно на редких событиях, где важнее не пропустить нужные кадры. Для задач с вероятностными ответами полезна и «логарифмическая потеря»: она штрафует особенно сильно за уверенные, но неправильные прогнозы, и тем самым поощряет честные вероятности.



Казалось бы, раз уж все завязано на вероятностях, нужна огромная модель и бесконечные данные. На практике почти всегда выигрыш

вает «обучение по переносу»: мы берем сеть, которую уже учили узнавать тысячи бытовых объектов на гигантской коллекции картинок, и доучиваем ее под нашу задачу. Такая сеть уже «понимает», как устроены формы и текстуры, и ей гораздо легче адаптироваться к кошкам именно в наших ракурсах или к дефектам именно на наших деталях. Это резко снижает потребность в аннотациях: сотни или тысячи правильно размеченных изображений часто дают рабочее решение, особенно если грамотно устроить валидацию и аугментации — искусственные преобразования исходных картинок, которые учат модель быть устойчивой к вариациям вроде поворотов, шумов и разного освещения.

У классификации есть свои типичные ловушки. Самая заметная — дисбаланс классов. Например, когда редкие события, ради которых все и затевается, представлены плохо, и модель привыкает угадывать ответ случайным образом. Здесь помогают несколько простых приемов. Во-первых, разделять данные на обучение и проверку так, чтобы доли каждого класса были близки и там, и там; иначе можно случайно «выучить» редкий класс только на валидации и сделать неверные выводы о качестве модели. Во-вторых, давать редким примерам больше «веса» при обучении или подбирать функцию ошибки, которая сильнее наказывает за промахи именно по ним. В-третьих, сознательно «перекручивать» данные: добавлять преобразования именно для редких классов или чуть чаще подмешивать их в мини-партии при обучении.

Вторая ловушка — ложные подсказки в фоне. На снимках с автобусами чаще оказывается дорога с разметкой; модель легко учится распознавать не автобус, а характерные полосы. На рентгенах с пневмонией могут встречаться отметки маркера, и сеть запоминает их форму вместо того, чтобы смотреть на легочные поля. Здесь очень важна проверка данных: просматривать случайные и «наиболее уверенные» ответы модели, выясняя, на что она смотрит; показывать объект в разных окружениях; специально собирать примеры-контрпримеры, где непредсказуемый фон «сломает» возможную хитрость модели. Полезен и простой тест: перемешать фон или слегка замаскировать центральный объект и посмотреть, насколько сильно падает уверенность.

Третья проблема — сдвиг домена. Модель училась на снимках днем, а в реальности работает ночью; разметку делали на одном сканере, а ставим мы систему на другой; фотографии для каталога приходили от профессионалов, а теперь их снимают пользователи. Внешне это «те же» изображения, но статистика пикселей и сюжетов изменилась, и качество проседает. Начните с аугментаций, которые имитируют новый мир: меняйте яркость и контраст, вводите шум, имитируйте компрессию. Если есть возможность, немного доучите модель на сдвинутых данных, хотя бы с грубой разметкой: это может частично реставрировать

качество модели. И обязательно измеряйте качество по сценариям отдельно: не довольствуйтесь одним измерением, когда у вас на самом деле разные условия съемки.

Вернемся к порогам. Модель возвращает вероятности, но решать-то надо бинарно: считать ли это «браком» или «нормой». Универсального порога не существует. Если на конвейере критично не пропустить ни одного дефекта, порог придется опустить и принять больше ложных тревог, переложив их на вторичную проверку человеком или более медленную систему. В медицине часто делают наоборот: держат высокий порог и используют модель как «подсказчика», который поднимает руку лишь при высокой уверенности. В многоклассовых сценариях пороги разумно подбирать по-отдельности: «закат» на фотографиях распознается надежнее, чем «счастье», и к ним будут разные требования.

Есть еще одна тонкость, редко обсуждаемая вне практики, — честность вероятностей. Мы хотим, чтобы «0,8» действительно означало «примерно восемь из десяти похожих случаев верны». Это важно, когда решения принимаются автоматически и последовательно: например, в системе модерации контента уровни уверенности могут определять, отправлять ли пост на ручную проверку или блокировать сразу. Если вероятность «раздута», система будет слишком агрессивной; если «зажата», наоборот, пропустит нежелательное. Простой способ поправить шкалу — «калибровка» на валидационных данных: мы подстраиваем преобразование выходов модели так, чтобы они лучше совпадали с реальными частотами событий. Это не делает модель умнее, но делает ее ответы полезнее.

У меня есть хороший пример и из личного опыта. Для каждого миллиардного удара мне нужен был классификатор его типа: «разбой пирамиды», «обычный удар». Я собрал корпус изображений именно под эту задачу: положительные кадры — готовая к разбою пирамида, отрицательные — свободно раскатанные шары без структурированного треугольника. Весь поток данных перед подачей в нейросеть я приводил к одному формату: сначала вырезал из общего ракурса стол по четырем углам и нормировал размер.

Модель делал бинарной: «пирамида есть» против «пирамиды нет». Аугментации: изменение яркости и контраста под реальные залы, тонкие цветовые сдвиги под теплые лампы и зеркальное отображение по горизонтали. Дополнительно я подмешивал «помехи» — кисти рук, фрагменты кия, тени на пирамиде — чтобы сеть не терялась, когда игрок заносит кий над треугольником. Отдельным классом это не размечал, просто расширил отрицательную корзину примерами «почти-пирамида», где несколько шаров смещены или не хватает вершины треугольника. Разумеется, обучение происходило не с нуля, применялся перенос

обучения. Это когда я использую уже обученную модель — извлекатель признаков и дообучаю ее на своих данных под свою задачу. В качестве извлекателя признаков я выбрал нейросеть YOLO11n.

Инференс встроил в цикл верификации удара. Для каждого кандидата на удар я брал по кадру «до» и «после» события, прогонял их через классификатор. Правило простое: если до удара «пирамида есть», а после «пирамиды нет», это разбой. Если до — «есть» и после — «есть», значит, игрок сделал холостой замах или подстройку, и система помечает эпизод как «ложный удар» и ждет реального разбоя. Если до — «нет» аи после — «нет», это обычный ход в партии. Порог я подбирал на валидации, отдельно контролируя ошибки первого рода: ложный разбой портит статистику начала партии сильнее, чем пропуск одного холостого замаха.



И, наконец, пара слов о проверках здравого смысла. Прежде чем учить большую модель, полезно убедиться, что данные живые: посмотреть случайные примеры и их метки, поискать дубликаты, убедиться, что сходные картинки не пролезли одновременно в обучение и проверку, посмотреть, не утекает ли простая метка в имя файла. Такие проверки часто спасают недели экспериментов: выясняется, что половина «дефектов» размечена как «норма», или что редкие классы делятся внутри на два подтипа с совершенно разными внешними признаками.

Если подытожить, классификация — это не только «сказать, что на картинке», это про проектирование системы принятия решений. Мы определяем, какие именно ответы нам нужны, учим модель давать вероятности, которые соответствуют реальному миру, и подбираем способ превратить эти вероятности в действия с учетом стоимости ошибок. В этом смысле хороший классификатор — это не только удачная сеть, но и аккуратно собранные данные, честная оценка качества и продуманные пороги, согласованные с задачей бизнеса или практики. Когда все эти части настроены, простая на вид задача становится надежным фундаментом, на котором строятся более сложные системы машинного зрения.

CNN детекция объектов: где именно и что именно

Если в классификации мы просим модель назвать «что на картинке вообще», то в детекции вопрос точнее: «сколько тут отдельных объектов, где каждый из них находится и к какому типу относится». Ответ — набор прямоугольников с подписями и степенью уверенности. На снимке перекрестка это могут быть автомобили, автобусы, пешеходы и светофоры; на конвейере — болты, шайбы и «пустое место»; в магазине — люди и тележки, в моем проекте — шары, лузы, кии. Прямоугольник — это практичный компромисс: он достаточно прост, чтобы его быстро считать, и достаточно выразителен, чтобы локализовать большинство предметов.

Разметка под такую задачу выглядит просто: человек-оператор рисует вокруг каждого видимого экземпляра объекта рамку и выбирает нужную подпись из словаря классов. Но в этой «простоте» уже спрятаны две важные тонкости. Первая — единые правила: как обводить частично скрытый предмет, нужно ли включать в рамку выступающие детали, что считать отдельным экземпляром, если предметы соприкасаются. Вторая — плотность: на складе или на многолюдной улице рамок будет очень много, и даже небольшие расхождения в их границах между аннотаторами приведут к шуму в данных. В хороших проектах всегда есть «памятка разметчика» с пограничными примерами и периодическая сверка согласованности: это экономит недели обучения и отладки.

Результат работы детектора оценивают не по одной-единственной цифре. Прежде всего интересует, насколько хорошо модель совпадает с разметкой по положению и размерам рамок. Для этого считают отношение доли площадей пересечения предсказанных и истинных рамок к площади их объединения. Эта величина — от нуля до единицы — отвечает на вопрос «насколько сильно наши прямоугольники накрывают один и тот же объект». Если доля пересечения выше порога, считаем, что попадание засчитано. Дальше возникает естественный вопрос: что считать «хорошим попаданием» — пересечение половины площадей, трети или почти полного совпадения? Чтобы не «подогнать» результат под один-единственный выбор, современные бенчмарки усредняют качество по нескольким порогам сразу: от расслабленных до строгих. Получается показатель, который одновременно учитывает и умение находить объекты, и аккуратность их обводки. Дополнительно смотрят на качество по размеру целей — маленькие, средние и большие объекты — и на способность «дотягиваться» до разных уровней полноты: насколько модель увеличивает долю найденных объектов, если разрешить ей выдавать больше срабатываний. В прикладных системах к этим цифрам

всегда добавляются показатели скорости: сколько кадров в секунду система успевает разобрать, какой у нее средний отклик на кадр, как меняется задержка в пиковые моменты.

Разные семейства детекторов решают задачу по-разному, и здесь важно понимать общий характер компромисса. Есть «двухшаговые» методы: они сначала предлагают небольшое число «кандидатских» областей, где может быть объект, а потом для каждой такой области отдельно уточняют границы и класс. Такой подход исторически показывал очень высокую аккуратность, особенно на сложных сценах, но за это приходилось платить скоростью. Примеры — R-CNN. «Одношаговые» методы сразу пытаются «покрыть» все изображение сеткой возможных мест и размеров рамок и выдать готовые ответы за один проход. За счет этого они быстрее и проще в эксплуатации; неслучайно именно они чаще оказываются в продакшене — от дорожных камер до подсчета людей на входе. Примеры — YOLO, SSD. Наконец, есть направления, которые уходят от заранее заданной сетки вариантов и опираются на более «объектно-центричную» логику: вместо перебора множества рамок модель учится предлагать немного кандидатов в нужных местах или вообще предсказывает фиксированное число «слотов» под объекты на каждом кадре. Такие решения уменьшают количество постобработки результатов и часто ведут себя стабильнее на загруженных сценах, где предметы перекрывают друг друга.

Один из самых важных этапов в детекции и пример постобработки — «примирение» конкурирующих рамок. Представьте себе витрину с фруктами: модель увидела яблоко и выдала пять очень похожих рамок вокруг него с близкими уверенными оценками. Нам не нужно пять яблок, нам нужно одно. Уже известная нам классическая процедура NMS в таких случаях оставляет самую уверенную рамку и «подавляет» все остальные, которые с ней слишком сильно пересекаются. На редких и не перегруженных сценах это прекрасно работает. Но стоит перейти к толпе людей в метро или к аккуратно выложенным батарейкам на полке, как рамки разных объектов начинают пересекаться почти так же сильно, как рамки одного и того же. Жесткое «подавление» начинает стремительно «съедать» соседние объекты. Здесь помогают более мягкие правила: вместо того чтобы выбрасывать конкурирующие рамки, мы просто снижаем порог их уверенности в зависимости от степени пересечения. В результате модель перестает мешать сама себе на плотных сценах, а действительно лишние срабатывания все равно уходят.

Есть и другая классическая боль — маленькие объекты. Пешеход на картинке, снятой с высоты, занимает десятки пикселей; болт на конвейере — лишь пару сотен. Если «смотреть» на такую картинку как на

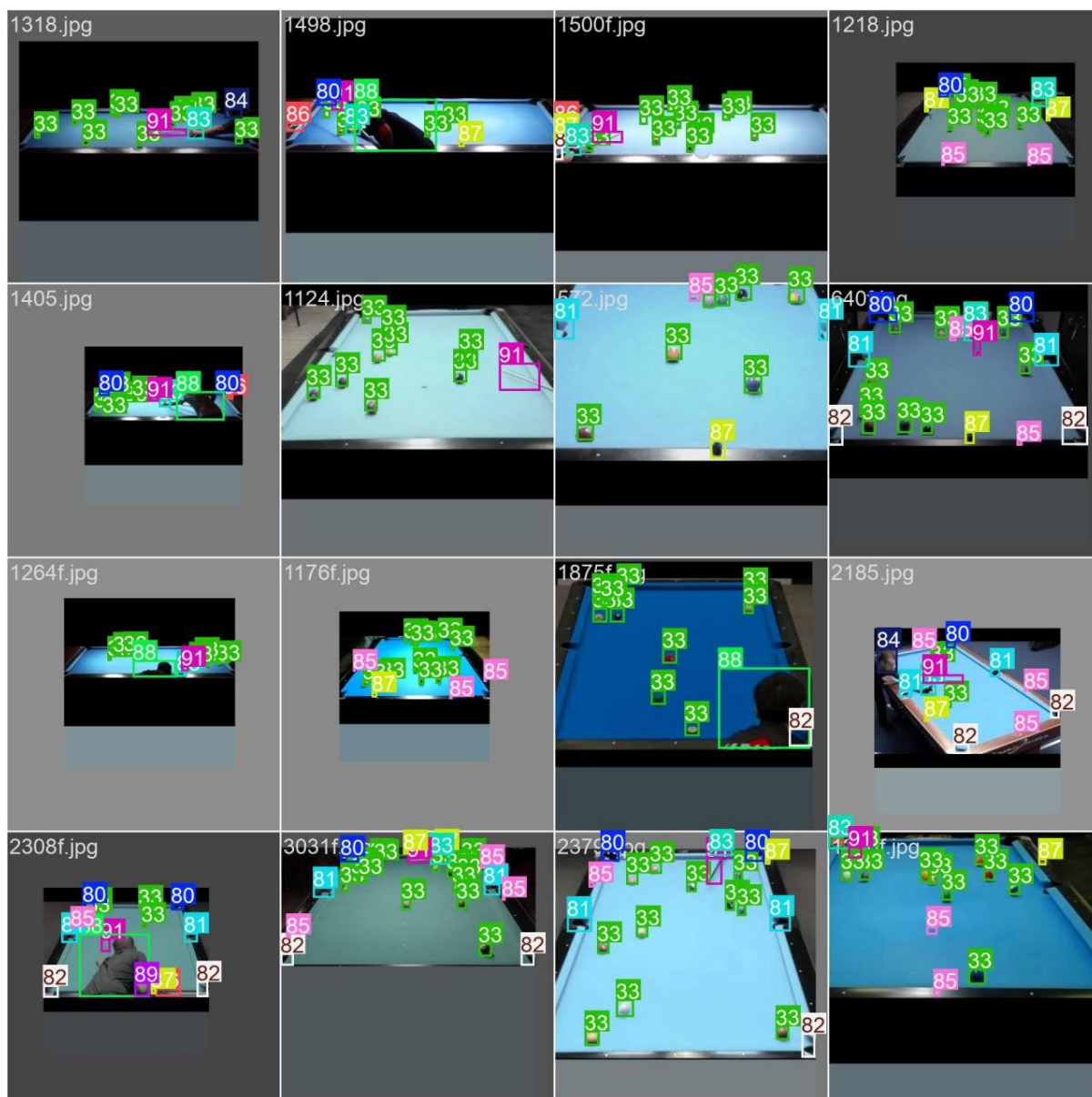
единое целое, эти крошечные детали просто тонут в общем шуме. Рецепт тут только один: заставить систему «подъезжать поближе» — подавать изображения в большем разрешении, разбивать кадр на плитки и анализировать их по отдельности, агрессивнее увеличивать маленькие объекты в ходе искусственных преобразований во время обучения. Помогает и разнообразие масштабов при обучении: когда одна и та же сцена показывается модели в разных размерах, она учится распознавать характерные контуры и текстуры даже тогда, когда объект в десять раз меньше, чем в учебном примере.

Третья частая проблема — частичные перекрытия. В реальном мире объекты редко бывают показаны целиком: люди заслоняют друг друга, шар наполовину уходит за ближний борт бильярдного стола, автомобиль прикрыт другим автомобилем. На таких сценах мы видим лишь фрагменты формы, и модель, если ее этому не учили, тянется к самым очевидным подсказкам: ярким надписям, ровным границам, привычному фону. Это ведет к пропускам и ложным срабатываниям. Тренировочные данные — наш главный инструмент защиты: стоит сознательно включать сцены с перекрытиями, вариативными ракурсами, сложным освещением, и система начинает опираться на более надежные, «глубокие» признаки. В оценке качества полезно отдельно смотреть на сцены «скученности», чтобы не терять эту проблему в общей статистике.

Наконец, длинный «хвост» редких классов. В приложениях наподобие розничной аналитики или инвентаризации часть объектов встречается очень часто, а часть — раз в неделю. Модель, стремясь минимизировать ошибки на большинстве, перестает «замечать» редкие. Это «лечится» увеличением выборок: можно немного переучивать модель на редких примерах, можно менять меру ошибки так, чтобы промахи по редким классам стоили дороже, можно ввести отдельные «мини-курсы» — короткие дообучения, целиком посвященные «хвосту». Важно и то, как вы тестируете систему: усредненная цифра по всем классам будет красивее, но ни о чем не скажет владельцу склада, для которого критичен именно редкий, но дорогой артикул.

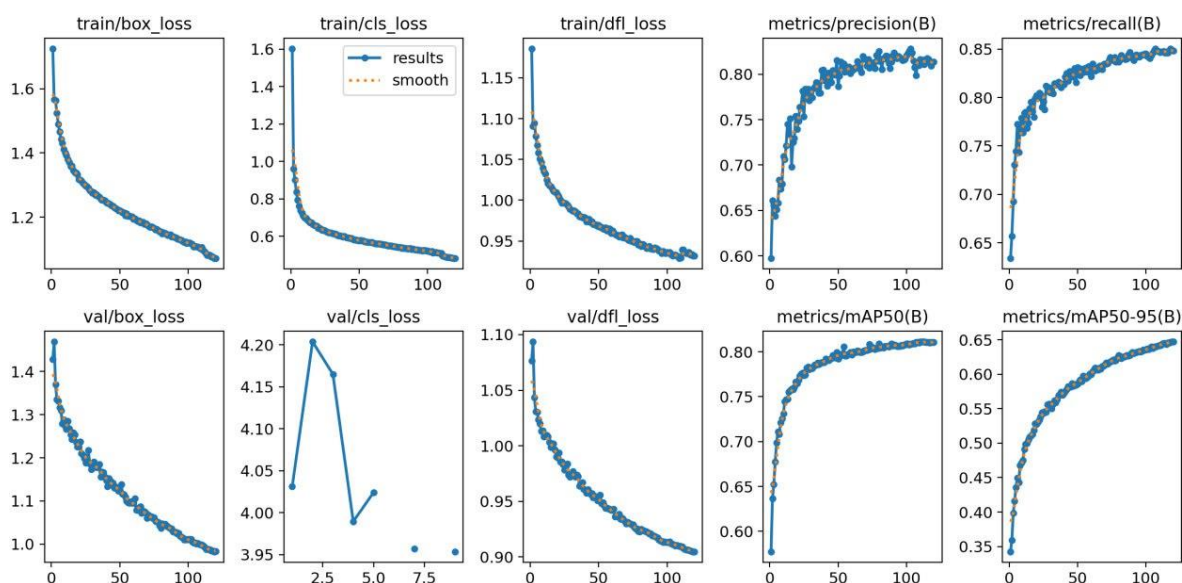
Чтобы почувствовать всю цепочку на практике, представьте магазин самообслуживания. На входе — счетчик людей: камера над дверью, поток из десятков лиц и капюшонов. Система должна мгновенно нарисовать рамки там, где есть человек, не перепутав отражения в стекле, и выдать число. Здесь важна скорость и аккуратное «примирение» рамок: если подавлять слишком агрессивно, толпа «схлопнется» в несколько человек; если слишком мягко — получите двойной счет. В зале — контроль полок: камера видит, сколько осталось батончиков в лотке. Объекты маленькие, похожие, плотно уложены. Потребуются увеличенное

разрешение, обучение на «нарезанных» плитках и более мягкие правила работы с пересечениями. На складе — проверка комплектности: в коробке должны лежать три вида деталей. Тут участвуют и редкие классы, и частичные перекрытия; модель должна не только находить рамки, но и честно оценивать свою уверенность, чтобы запустить вторичную проверку, если что-то «не нравится».



У меня детекция бильярдных шаров начинается с подготовки кадра. Вырезается прямоугольник стола: это резко уменьшает фоновые срабатывания и делает размер шара постоянным в пикселях, что полезно для порогов и для оценки качества. На вход нейросети идет нормализованный фрагмент нужного разрешения; баланс белого стабилизирую по белому шару и патчу сукна, чтобы цвет не «уплывал» между залами. Сама «голова» детекции — YOLOm, но с дообучением под плотные круглые объекты. Я добавляю еще отдельные классы мешающих

предметов — лузы, кии, руки — и пустые кадры без целей, чтобы модель не дорисовывала шары там, где их нет. В алгоритме подавления перекрытий использую мягкий режим: при близких шарах «жесткая» NMS с высоким порогом легко схлопывает соседние рамки. Это особенно заметно в областях кучности после разбоя, где небольшой сдвиг порога решает, будет два шара или один «супершар». Для кадров со смазанными динамичными шарами из-за короткой выдержки камеры я страхуюсь временным усреднением: беру медиану по нескольким соседним кадрам, чтобы случайный блик не превратился в «новый» объект. Порог уверенности у меня зависит от контекста. Я беру средний и отступаю от него 3 сигмы. Центр шара я уточняю внутри рамки простым центроидом по маске яркости, чтобы компенсировать легкий сдвиг рамки при бликах и тенях. Дальше этим центрам уже все равно, на какой камере снят матч: траектории и события считаются в координатах стола, а вся сложность света и оптики остается за порогом детектора. Именно так одна и та же схема корректно работает и на стандартной камере клуба, и на уличной съемке при солнечном свете — с теми же правилами и теми же порогами. На изображении ниже — типичная картина отчета по процессу обучения, которую создает штатный инструмент YOLO. Интуитивно понятно, что при хорошем обучении все метрики вписываются в некие экспоненциальные последовательности по росту точности, потерям и другим показателям.



Отдельного упоминания заслуживает эволюция самих подходов. Ранние системы полагались на перебор заранее заданных рамок разных размеров и форм: это делало их предсказуемыми и быстрыми в реализации, особенно на ограниченном железе. Позже появились архитектуры, которые учатся предлагать кандидатов «по месту», а не равномерно «накрывать» всю картинку. Еще дальше — модели, которые сразу

выдают конечный список объектов без последующей «чистки» перекрывающихся рамок: они формулируют задачу так, чтобы каждый предсказанный прямоугольник соответствовал какому-то реальному объекту или пустому месту. Эти сдвиги, это систематическая попытка убрать из пайплайна хрупкие эвристики и переложить больше ответственности на обучаемую часть модели. Это дает выигрыш в стабильности и простоте эксплуатации — выбор определяется требованиями к скорости, доступным оборудованием и сложностью сцен. Подробнее по разным подходам и архитектуры речь пойдет далее в хронологическом порядке их появления.

И, как всегда, на переднем плане — связь с бизнес-целями. Детектор с аккуратными рамками, но высокой задержкой может оказаться бесполезным для безопасности на производстве; сверхбыстрый, но «неряшливый» — приведет к лавине ложных тревог и выгоранию операторов. Поэтому на пилоте важно не только охотиться за каждым процентом интегральной метрики, но и смотреть глазами пользователя: как система ведет себя в час пик, что она делает на сложных сценах, как меняется ее уверенность при переключении камеры. В этом балансе — и инженерное мастерство, и искусство постановки задачи: детекция — это больше чем просто рисовать прямоугольники вокруг объектов. Важно всегда сопоставлять детекцию с её конечной задачей — получением результата, у которого есть цена ошибки и цена промедления.

CNN Сегментация: «раскрасить» сцену по смыслу

Если детекция отвечает на вопрос «где лежат объекты», то сегментация идет дальше и пытается «раскрасить» картинку так, чтобы каждый пиксель получил осмысленную роль. В самом простом варианте это похоже на детскую раскраску: небо — синим, дорога — серым, трава — зеленым. Такой режим называют семантическим: нас интересуют классы, а не конкретные экземпляры. На дорожной сцене все пешеходы попадут в один цвет «пешеход», а все машины — в цвет «авто». В другом режиме важна индивидуальность: нам нужно не просто понять, что на полке лежат батончики, а отделить один батончик от другого, даже если они соприкасаются. Тогда мы рисуем отдельную маску для каждого экземпляра — это и есть instance-сегментация. Наконец, в реальной жизни удобно сочетать оба взгляда: и знать, где на изображении «дорога» или «стена», и одновременно различать каждого человека или каждую коробку. Такой единый взгляд называют паноптическим: он объединяет семантическую «раскраску» и индивидуальные маски экземпляров.

Сегментация требует самой дорогой разметки. Нарисовать прямоугольник вокруг шара — секундное дело. Аккуратно обвести его контур, не залезая на фон, — минута, а то и больше на сложных кадрах. В медицине это часы: врач бережно обрисовывает опухоль по границе плотностей, а второй врач перепроверяет. И здесь особенно важны правила. Будем ли мы включать в маску тени? Как поступим с размытыми границами на фоне шумов? Считать ли крошечные островки дефекта частью дефекта или мусором? Хорошая разметка — это согласованность между разметчиками и четкие инструкции на пограничные случаи. На практике используют и «облегченные» подсказки: наброски линий и точек, из которых полуавтоматические инструменты восстанавливают аккуратную маску. Это сильно ускоряет процесс без жертв в качестве.

Как оценивают такое решение? Интуитивный ответ — «насколько хорошо совпали маски». Формально это отношение долей пересечения площадей предсказанной и истинной маски к их объединению. Если две фигуры идеально совпали, доля равна единице; если они почти не перекрываются, стремится к нулю. Обычно усредняют этот показатель по всем классам и кадрам: так мы получаем честную оценку общего качества. Для медицинских задач, где борются за каждую долю процента, часто используют и альтернативную меру «схожести», которая особенно чувствительна к пропускам и пересечениям — она удобна тем, что дает интуитивный масштаб: «насколько две фигуры похожи в среднем». В паноптическом режиме добавляется новая тонкость: важно не только «как покрашено», но и правильно ли сопоставлены экземпляры. Если у нас

два человека, а модель нарисовала одну большую маску на обоих, формально пикселей может совпасть немало, но смысл потерян. Для таких случаев используют метрику, которая одновременно учитывает и точность масок, и корректность «сопоставления персон». Наконец, есть задачи, где главное — не площадь, а границы: дорожная разметка, трещины на металле, сосудистые сети. Там полезно считать качество по контурам: «насколько аккуратно идут границы, не рвутся ли, не искажаются ли».

Сегментация используется в приложениях, где важна точность формы. В промышленной инспекции мы ищем волосковые царапины на полированной поверхности: прямоугольник вокруг детали ничем не поможет, нужна маска самого дефекта с точными краями, чтобы вычислить длину или площадь повреждения. В медицине картина схожая: площадь и форма очага на МРТ нужны не сами по себе, а как аргумент в клиническом решении — тут важнее пропустить как можно меньше «настоящих» пикселей опухоли, даже ценой небольшого захвата фона. В картографии сегментация выделяет здания и дороги на спутниковых снимках: от аккуратности зависят не только точность карты, но и правильные расчеты городской застройки. В робототехнике семантика дорожной сцены помогает разделить «где можно ехать» и «где нельзя», и тут на первый план выходят требования по скорости и устойчивости: роботу важно получить пусть и слегка шероховатую, но быструю маску свободного пространства, чем идеальную, но с опозданием.

У сегментации есть характерные «боли». Первая — тонкие структуры. Трещины, провода, сосуды, разметка на асфальте — все это легко теряется, если сеть «смотрит» на изображение слишком грубо или если мы сжимаем картинку перед обработкой. Вторая — дырки и «рванные» маски: модель улавливает общий контур, но внутри оставляет лакуны из-за бликов, текстур и слабых контрастов. Третья — огромный перевес фона. На типичном снимке металлургической заготовки нужные дефекты занимают доли процента, а все остальное — однообразный фон. Если обучать модель «как есть», она быстро учится всегда красить фон и почти никогда — дефект: формально среднее качество будет приличным, но пользы — ноль.

Универсальный совет заключается том, чтобы везде руководствоваться здравым смыслом. Во-первых, подгоняйте масштаб под задачу. Если результат определяется по тонким линиям, лучше обрабатывать изображения крупнее, даже ценой разбиения на «плитки» и склейки результата — это разрушает тонкие детали куда меньше, чем агрессивное сжатие. Во-вторых, учите модель уделять больше внимания редким пикселям. Достаточно настроить обучение так, чтобы ошибки по редким областям «звучали громче», чем промахи по вездесущему фону. Помогает

сознательно «перекармливать» сеть кадрами, где интересующий класс есть, и разнообразить их аугментациями — менять освещение, контраст, добавлять шумы, — чтобы модель перестала полагаться на случайные подсказки. В-третьих, не ленитесь делать постобработку результата. Простые операции вроде «зашить дырки» в маске, «сгладить» рваные края, почистить явно шумовые точки площадью в несколько пикселей — это разумный компромисс. Есть и более сложные приемы постобработки: подтягивать границы к сильным контрастам в исходном изображении, чтобы маска легла ровнее по реальной кромке объекта. Эти шаги можно настроить один раз под конкретную задачу и затем включить в конвейер.

Пара практических примеров, как это работает. Возьмем линию полировки алюминиевых дисков. Камера снимает блестящую поверхность, на которой царапины то видны, то теряются в бликах. Мы учим модель искать именно царапину как фигуру: вытянутую, тонкую, с неровной линией. Эталон — аккуратно обведенные мастером маски, где даже мелкие ответвления включены. На проверке нас интересует не только среднее совпадение площадей, но и «честность» границ: если модель пропустит тонкие дефекты, деталь с пограничным дефектом уйдет в «норму». Цена ошибки тут однозначна: пропуск — это риск для безопасности и репутации, а лишний брак — деньги. Поэтому пороги решения выставляют консервативно, а постобработку настраивают так, чтобы «зашивать» крошечные дыры, но не раздувать маску чрезмерно.

Теперь — МРТ головного мозга. Врач отмечает очаги, другой врач перепроверяет. Две маски редко совпадают идеально: границы размыты, разные специалисты по-разному трактуют переходные зоны. Мы учим модель на согласованных примерах и оцениваем ее по мере, которая особенно чувствительна к пропускам: от этого зависит, заметим ли мы изменения от обследования к обследованию. В интерфейсе врач видит не только маску, но и оценку уверенности по пикселям, чтобы быстро пробежаться по сомнительным местам. Любая «экономия» на разметке здесь выйдет боком.

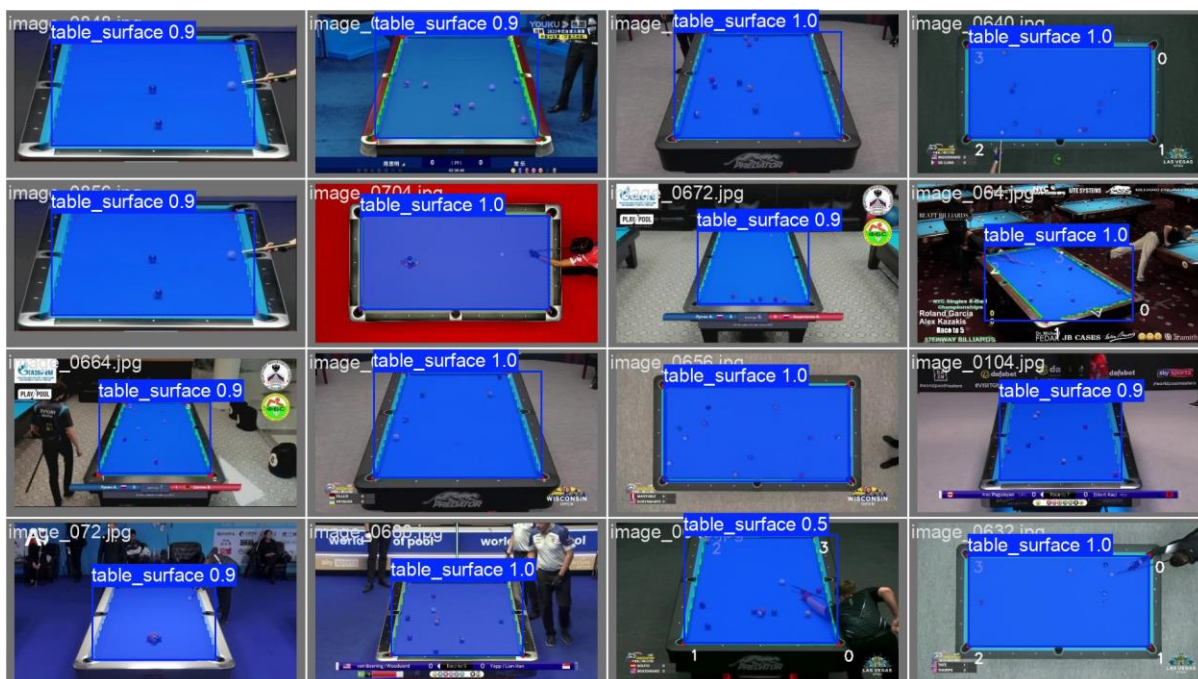
Еще пример — навигация для мобильного робота или автомобиля. Камера смотрит вперед, и каждую сотую долю секунды надо обновлять «карту занятости»: где свободно, где препятствие. Здесь паноптическая избыточность не нужна — важнее простая и быстрая семантика: пол и проезжая часть — один цвет, все остальное — другой. Ради скорости мы сознательно упрощаем модель и уменьшаем входное разрешение, но компенсируем это осторожной «плиточной» обработкой вокруг важных зон, куда робот собирается ехать. В проверке качества мы

держим в голове не только совпадение масок, но и время: решение, пришедшее на 200 миллисекунд позже, может быть хуже более грубого, но своевременного.

Практический пример из моего опыта. Задача была автоматизировать поиск игровой поверхности на кадре бильярдного матча. Как я уже писал ранее, у нас есть база данных детально размеченных бильярдных видео, включая случайное изображение с каждого ракурса, с отмеченными вручную углами игровой поверхности.



Далее мы подготовили датасет из 1407 таких изображений для обучения и 202 для валидации обучения. К каждому изображению идет файл аннотации в формате .txt где на одной строке через запятую приведены нормализованные координаты каждой точки в виде: $n, x_1, y_1, x_2, y_2, \dots, x_n, y_n$ (0 0.8708333333333333 0.687962962963 0.075 0.7101851851851851 0.3291666666666666 0.1583333333333333 0.6708333333333333 0.1398148148148148). Для ускорения и повышения точности обучения применили метод переноса обучения — использовали уже обученную нейросеть сегментатор YOLO11n_seg, созданную с помощью извлекателя признаков YOLO11n на датасете MS COCO с сегментацией объектов.



В результате новая нейросеть начала самостоятельно находить углы стола на новых видео, без участия человека. Точность нахождения не всегда получается достаточной, поэтому приходится делать сложную постобработку — сводить весь набор полученных точек к четырем базовым углам, усреднять координаты по серии сегментаций, выбирать поверхность с максимальной площадью. Также по полученному полигону мы формируем прямоугольную рамку стола для грубой обрезки кадра, для дальнейшего трекинга.

Всю разметку мы делаем своими собственными веб инструментами. Но сейчас уже вырос целый класс общих инструментов, делающих сегментацию более удобной. Это и семейства архитектур, которые удачно сочетают «видение деталей» с «пониманием общего контекста», и интерактивные методы, где человеку достаточно поставить несколько «плюсиков» и «минусиков», а система сама дорисует маску вдоль реальных границ объекта. Такие инструменты экономят часы разметчика и позволяют быстро запускать пилоты там, где раньше требовались месяцы кропотливой работы.

В конечном счете сегментация — это про форму как смысл. Мы не просто узнаем «что это такое», а буквально измеряем мир: площади игровой поверхности, длину трещин, долю свободного пространства и очертания зданий. Именно поэтому к ней предъявляют две пары требований: точность масок и уместность по скорости. Баланс между ними всегда задает практическое приложение: где-то важнее «не упустить ни пикселя» и принять ручную доработку, где-то — «ответить быстро и стабильно».

CAFFE

В начале 2010-х годов, когда сверточные нейронные сети только начинали выходить из тени лабораторных экспериментов, чтобы стать двигателем прогресса в компьютерном зрении, фреймворк Caffe — Convolutional Architecture for Fast Feature Embedding — стал одним из первых мостов между академической теорией и практическим внедрением. Он был разработан в 2013 году командой Berkeley Artificial Intelligence Research (BAIR) под руководством Янцин Цзя. Среди разработчиков были: Эван Шелхаммер, Джефф Донахью, Сергей Караев, Джонатан Лонг, Росс Гиршик, Серхио Гуадарама, Тревор Даррелл. Также в разработку Caffe внесли вклад другие члены команды, среди них: Карл Доерш, Эрик Тзэн, Эван Шелхаммер, Филипп Крахенбюль, Ронгханг Ху, Росс Гиршик, Такуя Нарихира и другие. Этот инструмент, изначально ориентированный на ускоренную обработку изображений, предлагал предельно лаконичный синтаксис на основе протокольных буферов (protobuf) для описания архитектур нейросетей, что позволяло быстро собирать по блокам сложные модели — от простых сверток до многослойных CNN вроде AlexNet — в текстовом формате, автоматически компилируемом в C++ для максимальной производительности. Ключевым преимуществом стала ранняя интеграция с CUDA, обеспечивавшая параллельные вычисления на видеокарте NVIDIA, где базовые операции вроде сверток и пулинга выполнялись в 10–20 раз быстрее, делая обучение на датасетах вроде ImageNet реальностью в домашних условиях.

Caffe быстро завоевал популярность среди исследователей: его модульность позволяла быстро экспериментировать с комбинациями слоев без углубления в их исходный код. Плюс были встроенные инструменты для визуализации и отладки, такие как `draw_net.py`, которые превращали абстрактные графики в наглядные диаграммы, что особенно ценилось в эпоху, когда глубокое обучение еще не имело унифицированных экосистем. Этот фреймворк стал основой для сотен публикаций на CVPR и ICCV, от ранних работ по детекции лиц до сегментации сцен, и даже повлиял на коммерческие продукты, такие как системы распознавания в Google Photos; однако к середине 2010-х его жесткая структура уступила место более гибким альтернативам вроде TensorFlow и PyTorch, где динамические графы и императивное программирование открыли двери для сложных задач вроде рекуррентных (RNN) и генеративно-сопоставительных нейросетей (GAN).

К слову, один из соавторов Caffe, Сергей Караев — заметная фигура в эволюции компьютерного зрения, родился в России и продолжил образование в США, где в 2014 году защитил докторскую диссертацию в Калифорнийском университете в Беркли под руководством Трево́ра

Даррелла. Он является автором научной работы **Recognizing Image Style** (с соавторами из Калифорнийского университета и компании Adobe), где изучалось распознавание «стиля» изображения и сравнивались наборы признаков, и линия про «anytime recognition» — распознавание с управляемой стоимостью вычислений. Он также соавтор системы **DenseNet** (не путать с DenseNet-архитектурой 2017 года, а ранняя система для быстрых пирамид признаков сверточной сети). В прикладной плоскости Караев — сооснователь сервиса **Gradescope** (2014), позже купленного Turnitin.

На этапе перехода к нейросетевым методам детекции бильярдных шаров на игровой поверхности мой выбор пал на фреймворк Caffe, несмотря на его солидный возраст — к тому моменту ему исполнилось уже более десяти лет, а обновления прекратились еще в 2017 году. Эта "старина" привлекла меня прежде всего возможностью параллельного обучения на нескольких GPU, что было критично для обработки большого датасета с тысячами аннотированных кадров из матчей. В результате тщательных экспериментов по серверному администрированию мы успешно задействовали пару NVIDIA GTX 1080 Ti в кластере под Ubuntu 20.04, самостоятельно собрав версию Caffe с полной поддержкой CUDA 10.2 и cuDNN 7.6, адаптированную для нативных моделей VGGNet-16. Этот процесс оказался технически изнурительным: пришлось вручную патчить устаревшие зависимости, такие как protobuf 2.6 и OpenBLAS, для совместимости с современными пакетами вроде GCC 9 и glibc 2.31, а также бороться с конфликтами в сборке из-за отсутствия официальной поддержки новых драйверов NVIDIA. Попытки портировать под Windows с Visual Studio провалились из-за нестабильности GPU-библиотек, так что Ubuntu стала оптимальным выбором. Параллельно мы обновили OpenCV с версии 3.4 до 4.7, собрав ее с флагами CUDA и contrib-модулями, что открыло доступ к новым функциям вроде DNN-модуля для инференса на GPU, улучшило стабильность в многопоточной среде и оптимизировало потребление памяти — теперь модели загружались в видеопамять более крупными пакетами, что сильно ускорило процесс обучения. Это позволило вернуться к знакомым пайплайнам на C++ с VGGNet/Caffe в OpenCV, где раньше обучение одной модели растягивалось на дни в CPU-режиме, а теперь укладывалось в 4–6 часов на двух картах, с ускорением детекции до 50 мс на кадр. В итоге первоначальный план миграции на TensorFlow или PyTorch был отложен: Caffe, с его предсказуемостью, идеально вписался в проект. Для меня тогда по практической отдаче проверенный инструмент, адаптированный под железо, казался более привлекательным, чем модульные новинки.

MS COCO

Очень быстро всем стало ясно: одной лишь классификации изображений недостаточно для настоящего понимания визуального мира — нужны инструменты, способные разбирать сложные, многокомпонентные сцены с учетом контекста и взаимодействий. В 2014 году эта потребность материализовалась в MS COCO — Microsoft Common Objects in Context, — грандиозном датасете, ориентированном на «обыденные» реальные сцены, где на одном снимке сосуществуют десятки объектов в естественном беспорядке, сопровождаемые не только метками, но и подробными подписями на естественном языке, пиксельными масками для семантической сегментации и ключевыми точками на человеческих фигурах для оценки поз. Этот набор включал около 330 тысяч изображений высокого разрешения, аннотированных более чем полутора миллионами экземпляров объектов из 80 категорий — от повседневных предметов вроде чашек и стульев до динамичных элементов вроде людей в движении, — с пятью уникальными подписями для каждой фотографии, сгенерированными через краудсорсинг для разнообразия формулировок. Задачи машинного зрения усложнились: помимо базового вопроса «что здесь изображено?», возникли более сложные вызовы — «сколько объектов присутствует, где они расположены с точностью до пикселя, какой формы и размера каждый из них, а также что именно делают люди в этой сцене?».

COCO радикально поднял планку понимания сцены, расширив исследовательскую повестку: семантическая и инстанс-сегментация, многообъектная детекция с учетом перекрытий, генерация описаний изображений на основе визуально-лингвистических моделей, оценка ключевых точек для анализа поз — все эти направления, ранее остававшиеся нишевыми экспериментами, ворвались в мейнстрим, став неотъемлемой частью ежегодных конкурсов вроде COCO Challenge и бенчмарков для новых архитектур. Для разработчиков алгоритмов это стало очередным вызовом: впечатляющая точность моделей, отточенных на изолированных, аккуратно обрезанных кропах из ImageNet, никуда не делась, но в хаотичных, многоплановых контекстах COCO они внезапно обнажили свои слабости — путаницу в плотных композициях, неспособность разделять соприкасающиеся или частично скрытые объекты, а также неустойчивость в условиях разнообразия реального мира с его вариациями освещения, углов и культурных нюансов. В итоге этот датасет не просто обогатил инструментарий, но и подтолкнул к эволюции подходов: от простых классификаторов изображений к многоуровневым системам, способным к контекстуальному рассуждению.

Если говорить про обучение моделей, то все начинается со стратегии сбора данных. Хочу поделиться с вами личным опытом практической работы с большими данными. Эта работа у нас началась с формирования базы данных подходящих видео. Мы изначально ограничили выбор только матчами новее 2019 года. Для каждого бильярдного матча мы размечали контуры игровой поверхности стола. Затем мы проходили всю запись по тайм-линии, через собственный веб интерфейс, специально созданный для этой задачи, отмечая начало и конец каждого удара вручную. Критерий старта — кадр, который предшествует началу движения битка; критерий завершения — стабилизация сцены после остановки шаров. Метки ставили в номерах кадров. Для каждого эпизода я сохранял изображение состояния стола до удара.

Одновременно с метками кадров я присваивал удару тип и результат. Чтобы нейросеть была переносимой между турнирами и залами, я сразу нормировал геометрию: для каждого эпизода вместе с временными отметками «начало/конец» в автоматическом режиме писал служебные поля — идентификатор стола и камеры, частоту кадров, координаты углов стола, а также идентификаторы игроков (если они известны) для привязки к ge-ID. В базу уходили и агрегаты по динамике: облако точек, число активных треков в первые N кадров, суммарная длина перемещений, оценка вращения шара, факт касания борта и другое — эти числа помогают автоматизировать валидацию (например, отличить реальный разбой от ложного движения перед ним) и служат слабыми признаками для других моделей.

Так, в результате систематической работы получилось собрать сотни тысяч ударов с четкими границами, типологией и исходами. Этот массив стал «опорным слоем» для всех последующих выборок: из тех же эпизодов я формировал корпуса для детекции (кадры «до» и «после»), для трекинга (полные временные окна с траекториями), для поведенческой классификации (укороченные нормированные последовательности), для переидентификации игроков (отрезки с игроком у борта), для оценки результатов удара (примеры «забито /промах») и прочее. В итоге любая новая аналитическая задача, требующая обучения специальной нейросети, решается подключением к уже готовой «фабрике» данных: формулируешь задачу, выбираешь архитектуру, задаешь фильтры для поиска данных, получаешь воспроизводимую выборку с одними и теми же нормализованными данными.

В 2021 году я зарегистрировал в Роспатенте Базу данных «Статистика ключевых параметров бильярдных ударов» с присвоением номера свидетельства 2021621586. База данных в формате коллекции json объектов состоит из эпизодов ударов в привязке к конкретному видео. Подготовка датасета для обучения детектора конкретно бильярдных шаров

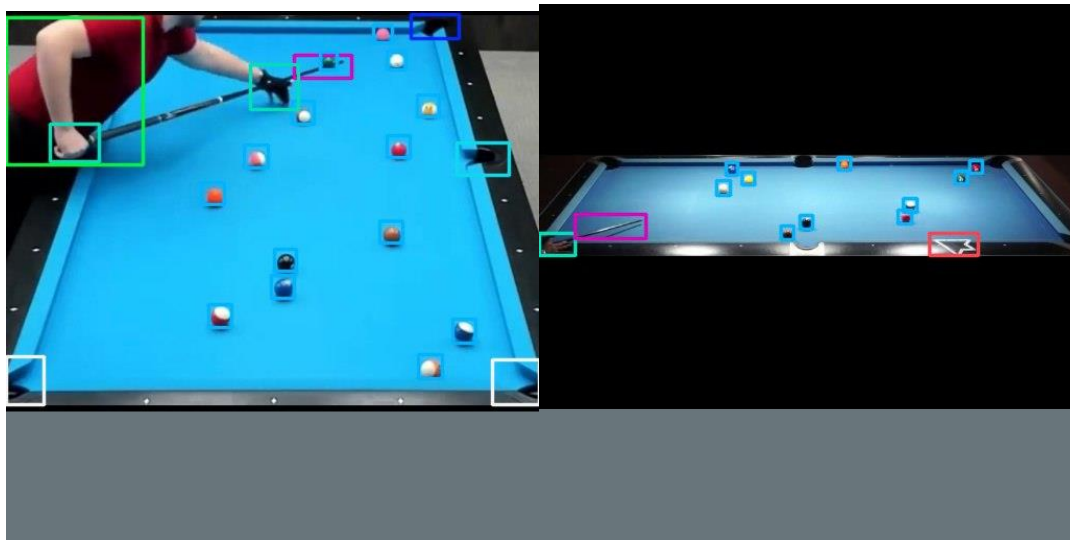
началась в том же 2021 году на базе датасета MS COCO. В этом датасете уже есть соответствующий класс под номером 32 - спортивные мячи, в который входят и мячи, и шары из разных видов спорта. Применительно к моему проекту, этот класс давал недостаточную точность. Для подготовки обучающей выборки мы сознательно собирали кадры не из одной записи, а из десятков турниров из открытых источников — разные страны, разные клубы, разный свет, разные цвета сукна и шаров. Камера во всех случаях статична, чтобы исключить паразитические эффекты от дрожания, но в остальном мы стремились к максимальному разнообразию условий. В итоге датасет для детекции шаров — это около 60 000 размеченных рамок шаров на около 8 000 изображений, извлеченных из 404 видео; на одном кадре могло быть до пятнадцати шаров. Такая плотность аннотаций делает датасет информативным при умеренном объеме изображений.

Для каждого видео мы сначала вручную выделяли контур бильярдного стола, потом автоматически вырезали изображение по этому контуру, приводили его к опорному масштабу (длинная сторона — 512 пикселей при сохранении пропорций) и укладывали на нейтральный квадратный фон фиксированным образом. Из всего матча нас интересовали наиболее репрезентативные кадры отдельных эпизодов. Мы брали второй удар в каждой партии с отступом около секунды от начала удара — так на столе одновременно оказывалось больше шаров, они надежно оказывались на разных позициях и часть из них была в движении. Это снижало риск дублирования одной и той же позиции одного и того же шара, что улучшало обобщающую способность модели.

Каждый видимый шар размечался прямоугольником произвольного размера, с двумя обязательными условиями: шар должен находиться внутри рамки целиком и быть максимально центрирован, а после первой проходки все изображения пересматривались повторно — наличие на обучающем наборе «забытых» шаров не допускалось. Такая дошная, но настойчивая проверка дает непропорционально большой выигрыш в точности. За организацию этой скрупулезной работы я выражаю отдельную благодарность Екатерине Корниенко. Дело стало возможным только благодаря ее природному вниманию к мелочам и огромной усидчивости.

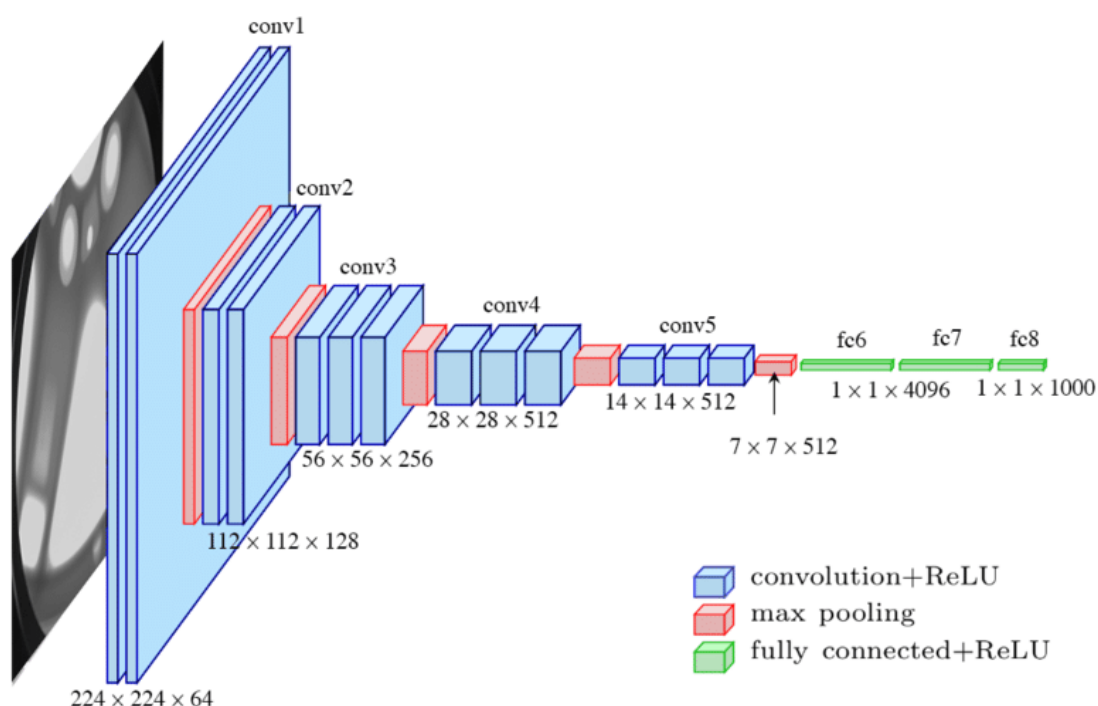
Как только базовая детекция «научилась» видеть то, что нужно, встал вопрос ложных срабатываний. Решение оказалось простым и рабочим: расширить количество классов и добавить нейтральные изображения без единого объекта. Помимо шаров мы ввели десять дополнительных классов часто встречающихся в бильярдном матче объектов — лузы, кии, отметки и предметы на столе, части тела игроков — и специально добавили несколько сотен нейтральных изображений без единого

целевого объекта. После переобучения на двух тысячах изображений с расширенной разметкой модель стала заметно лучше отбрасывать «шумы», а впоследствии весь набор был доведен до около восьми тысяч изображений уже с горизонтальным отражением как единственной аугментацией. По внутренней метрике детектирования прирост от удвоения набора составил порядка четверти, хотя важно помнить, что это «техническое» удвоение за счет отражения, а не за счет новых сцен.



Во внутренних экспериментах мы сознательно фиксировали, на каких именно классах учимся и как считаем итоговую оценку. Например, когда сводная метрика детектирования считалась по всему количеству классов (81) базовой модели, как на конкурсе, а реальных классов в нашей разметке было одиннадцать, ее абсолютные значения неизбежно отличались от классических — это нормальный эффект несоответствия метрик и его надо понимать.

VGGNet



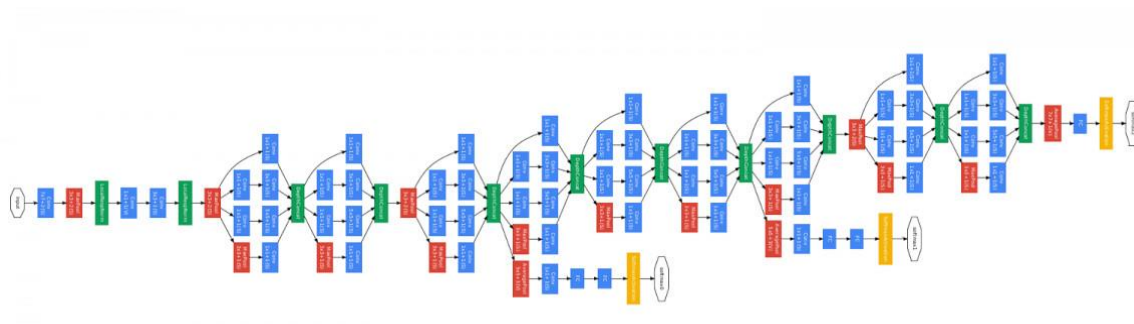
В 2014 году, на фоне триумфа глубокого обучения в рамках ILSVRC, оформились два ключевых архитектурных направления, каждое из которых одержало победу на конкурсе, но с принципиально разными подходами, подчеркивающими дилемму между простотой и эффективностью. Оксфордская группа Visual Geometry Group (VGG) продемонстрировала, что радикальное углубление сети может стать универсальным решением: их модели строились из однородных «кирпичиков» — маленьких сверток 3×3 с шагом 1, повторяющихся последовательно, — что позволило достичь 16–19 слоев с весами, без сложных ветвлений или инноваций в топологии. Эта стратегия, воплощенная в вариантах VGG-16 и VGG-19, обеспечивала интуитивную интерпретируемость: каждый слой последовательно извлекал все более абстрактные признаки, от краев до объектов, — но цена была высока — сотни миллионов параметров, что приводило к моделям весом в сотни мегабайт и требовало значительных ресурсов для обучения и даже инференса на GPU того времени. Тем не менее, VGG завоевала сердца сообщества своей простотой и воспроизводимостью: код легко масштабировался, а дизайн стал эталоном для бенчмарков. Она и по сей день остается основой для многих прототипов, включая анализ траекторий в динамичных сценах вроде бильярдных матчей.

Я уже упоминал выше, что вначале сам использовал именно модель VGG в качестве извлекателя признаков для детектирования бильярдных шаров в связке со слоями SSD, до которых мы еще дойдем, обученную на датасете VOC2007. Могу сказать, что в целом результат ее работы вполне удовлетворительный, даже спустя десять лет после ее открытия. Если не учитывать вопрос скорости работы, то это прекрасный старт для создания быстрого прототипа и отладки других элементов общего конвейера данных.

Если говорить о ключевой фигуре оксфордской группы VGG, то это, конечно, Эндрю Зиссерман. Он — профессор компьютерного зрения в Оксфорде и «профессор-исследователь Королевского общества», пришёл из классического компьютерного зрения, из геометрии изображений и многовидовой реконструкции, а затем стал одним из тех, кто не просто принял глубокие нейросети, а активно способствовал их внедрению в практику машинного зрения. Его книга с Ричардом Хартли «Многовидовая геометрия в компьютерном зрении» (Multiple View Geometry in Computer Vision) давно стала классикой, а награда «Королевского общества» в 2022 году отдельно подчёркивала, что он был пионером и в геометрическом анализе изображений, и в машинном обучении.

Карен Симоньян — соавтор VGG. На момент этой работы он был молодым исследователем. Позже он ушёл в DeepMind и уже там участвовал в работах совсем другого масштаба, например, в AlphaGo Zero. Если чуть расширять круг участников VGG-проекта, то очень заметен Андреа Ведальди. Именно через Ведальди и Зиссермана VGG стала поставщиком инструментария: учебных практикумов, кода, библиотек вроде MatConvNet, аннотационных инструментов и аккуратной инженерной культуры вокруг моделей. Это одна из причин, почему группа так сильно повлияла на развитие машинного зрения: они не просто публиковали идеи, а делали их воспроизводимыми и удобными для других. Даже на официальном практикуме VGG по CNN авторами указаны Ведальди и Зиссермана.

GoogLeNet (Inception)



Вторым архитектурным направлением, победившем на ILSVRC 2014 стала команда Google, ведомая Кристианом Сегеди, которая представила Inception (буквально — «зарождение», «начало», «внедрение идеи», известную также как GoogLeNet) — экономичную, многоканальную архитектуру, где каждый модуль работал как параллельный конвейер: внутри него одновременно применялись свертки разных размеров (1×1 для компактного сжатия каналов, 3×3 и 5×5 для захвата локальных и средних паттернов, плюс пулинг для глобальных), а избыточная размерность оперативно снижалась через «дешевые» 1×1 -свертки, минимизируя вычисления без потери выразительности. Эта дерзкая идея — не просто «глубже», но «умнее внутри», с акцентом на факторизацию операций и глобальный пулинг вместо плотных полносвязных слоев — позволила GoogLeNet достичь 22 слоев при всего 7 миллионах параметров, что сделало ее чемпионом ILSVRC-2014 с ошибкой в 6,7% по top-5, опередив VGG по эффективности на 30–40% в плане FLOPs (операций с плавающей запятой).

В итоге 2014 год четко разграничил две плодотворные линии развития: либо инвестировать в простоту за счет расточительного расхода памяти и вычислений (как в VGG, идеальной для исследований с обильными данными), либо полагаться на архитектурную изобретательность для достижения той же производительности при скромных ресурсах (как в Inception, предвосхитившей модульные дизайны современных сетей). Обе парадигмы не только выжили, но и эволюционировали, скрещиваясь с последующими инновациями — от ResNet с остаточными связями до EfficientNet.

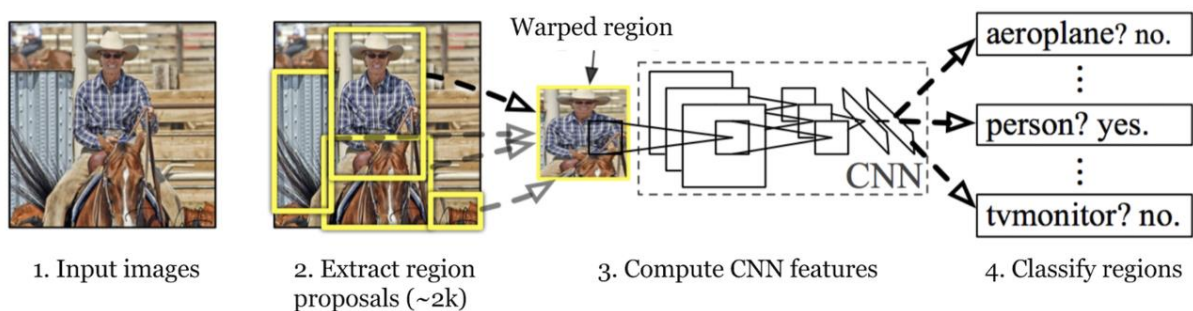
GAN - Генеративные нейронные сети

В том же 2014 году, когда COCO и новые архитектуры вроде VGG и Inception расширяли горизонты распознавания и понимания образов, параллельно развернулась ещё одна, не менее революционная ветвь развития машинного зрения — генеративная, где акцент сместился с анализа существующих изображений на создание новых. Иан Гудфеллоу, тогда аспирант Университета Монреаля, вместе с коллегами — Алексом Радфордом, Мехди Мирзой и Йошуа Бенжио — опубликовал статью «Generative Adversarial Nets» в NIPS, изобрел новый подход: генеративно-сопоставительные сети (GAN, Generative Adversarial Networks). В их основе лежит элегантная антагонистическая динамика, напоминающая нулевую сумму в игре: генератор — «художник» — учится синтезировать реалистичные образцы из случайного шума, стремясь обмануть дискриминатор — «критика», который, в свою очередь, оттачивает способность отличать подлинные данные от фальшивок; эти две сети обучаются одновременно в минимизационно-максимизационной петле, где улучшения одного провоцируют эволюцию другого, пока равновесие не достигнет уровня, где синтез становится неотличим от реальности. Первые эксперименты, проведенные на простых датасетах вроде MNIST (цифры) и CIFAR-10 (маленькие цветные изображения), производили скромные результаты — размытые, лица или объекты низкого разрешения, далекие от фотореализма, с артефактами вроде шумов и искажений, — но сама концепция «обучения через соперничество» мгновенно зажгла интерес, поскольку решала фундаментальную проблему генеративных моделей: отсутствие явного учителя для оценки качества. Уже к 2015–2016 годам GAN эволюционировали в мощные инструменты — от DCGAN с сверточными слоями для высококачественной генерации до CycleGAN для переноса стилей без парных данных, — позволив создавать гиперреалистичные лица несуществующих людей (как в StyleGAN), дорисовывать недостающие элементы сцен, стилизовать фото под картины Ван Гога или даже симулировать вариации освещения в видео. На картинке результат работы StyleGAN — оригинальные лица: верхняя строчка и левая колонка. Между ними - смешение стилей, генерация не существующих изображений.



Для 2014 это стало настоящим шоком: машинное зрение, ранее сосредоточенное на пассивном «узнавании» и классификации, внезапно обрело творческий потенциал — способность «создавать» миры, что открыло двери для аугментации данных в датасетах вроде ImageNet или COCO, где синтетические изображения помогали компенсировать редкие сценарии, такие как окклюзии или нестандартные ракурсы.

R-CNN



В целом 2014 год выдался обильным на открытия. Когда генеративные модели вроде GAN открывали новые горизонты творчества в машинном зрении, набирал силу прорыв в области детекции объектов. Исследователь Росс Гиршик из Microsoft Research представил Regions with CNN features (R-CNN), революционный подход, который радикально повысил точность обнаружения: модель извлекала признаки с помощью предобученной сверточной сети (например, AlexNet или VGG), применяя их к тысячам кандидатных регионов, предложенных отдельным алгоритмом вроде selective search, а затем классифицируя каждый с помощью SVM и уточняя границы через регрессию рамок. Качество взлетело — средняя точность (mAP) на PASCAL VOC достигла 53 процентов против предыдущих 30–35, — но цена была высока: обработка одного изображения занимала до 40 секунд на CPU из-за многоэтапного конвейера, делая систему непригодной для реального времени.

Уже в 2015 году та же команда Microsoft Research, включая Гиршика, довела концепцию до совершенства: Fast R-CNN оптимизировала пайплайн, заменив SVM на сквозную обучаемую сеть с многозадачной функцией потерь (классификация + регрессия границ), что ускорило обучение в 200 раз и инференс в 9 раз, достигнув 213 изображений в час на GPU. Кульминацией стал следующий Faster R-CNN, где узкое место — медленный внешний генератор регионов — был устранен: вместо него ввели Region Proposal Network (RPN), специализированную сверточную «сетку предложений», интегрированную в основную сеть, которая на основе тех же промежуточных признаков генерирует 200–300 кандидатов за миллисекунды, с анкер-окнами для учета масштабов и аспектов, и отсеивает нерелевантные через NMS.

К тому времени детекторы объектов эволюционировали к реальному времени на GPU — около 5–10 кадров в секунду: единая, сложенная архитектура училась одновременно «где искать» (предложения регионов) и «что именно там» (классификация), минимизируя потерю информации на стыках компонентов. Показательно, что варианты Faster R-CNN лидировали в треках детекции конкурсе ILSVRC 2015 (mAP 0,55) и COCO-2015 (mAP 0,37), подтвердив превосходство интегрированных

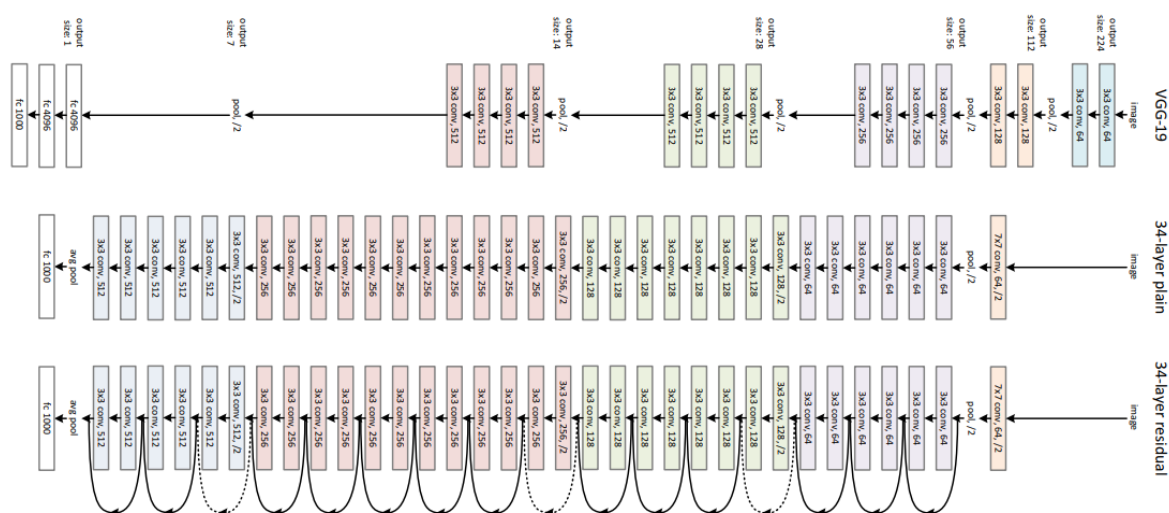
систем над конвейерами из несогласованных блоков: согласованность не только повышала точность на 10–15 процентов, но и упрощала развертывание.

Что касается дальнейшей эволюции семейства R-CNN, то в 2017 году команда Facebook AI Research — Каймин Хэ, Джорджия Гьоксари, Петр Доллар и Росс Гиршик — представила Mask R-CNN, элегантно расширив Faster R-CNN простым, но мощным дополнением: к существующим ветвям для классификации объектов и регрессии координат рамок добавили третью — полностью сверточную ветвь, генерирующую пиксельную маску для каждого обнаруженного экземпляра. Эта маска, предсказываемая как бинарная карта (foreground/background) в пространстве ROI (области интереса) с последующим изменением размера до оригинального размера изображения, позволяла не только локализовать объект, но и точно очерчивать его контуры на уровне отдельных пикселей, с минимальными дополнительными вычислениями — всего 2,9 миллиарда FLOPs на инференс по сравнению с 152 миллиардами у базового Faster R-CNN. В результате «трехветвенная» архитектура, опирающаяся на общий предобученный извлекатель признаков вроде ResNet-50 или ResNet-101, стала де-факто стандартом для сегментации объектов: она обеспечивала высокую точность (AP 37,1% на COCO для масок) при сохранении совместимости с задачами детекции, делая переход от рамок к пиксельным границам почти бесшовным. В практических приложениях — от медицинской визуализации, где маскирование опухолей на МРТ требует субмиллиметровой точности, до автономного вождения, где алгоритм должен не просто идентифицировать пешехода, но и точно сегментировать его силуэт для расчета траектории объезда, или робототехники, где анализ взаимодействий объектов (например, захват манипулятором) зависит от чистых контуров без перекрытий.

Следующая ветка — Cascade R-CNN превратила детектор в каскад из нескольких последовательно работающих ступеней. Первая ступень обучается на более мягком критерии перекрытия и выдаёт грубые, но правдоподобные гипотезы. Вторая получает уже улучшенные рамки и учится быть строже. Третья работает ещё избирательнее и доводит локализацию до более высокого качества. В классической конфигурации это выглядело как последовательность порогов перекрытия $0,5 \rightarrow 0,6 \rightarrow 0,7$: каждая следующая ступень училась на более аккуратных примерах, но брала их не из исходного хаоса, а из уже подправленного выхода предыдущего детектора. За счёт этого удалось одновременно избежать двух проблем: нехватки положительных примеров на высоких порогах и несоответствия между тем, на каких гипотезах обучался детектор, и тем, какие рамки он видит во время работы.

Практический смысл этой архитектуры в том, что она сместила фокус внимания к вопросу «насколько качественно мы локализовали объект». Cascade R-CNN сделала акцент на пространственной локализации за счёт такого подхода к обучению, чтобы каждая следующая ступень получала гипотезы именно того уровня аккуратности, под который она сама оптимизирована. Это сделало модель особенно сильной в задачах, где важна именно точная рамка: плотные сцены, мелкие объекты, строгие метрики качества, где небрежная рамка уже считается ошибкой. На наборе COCO такая схема превзошла прежние одноэтапные и обычные двухэтапные детекторы, а позже была естественным образом распространена и на сегментацию отдельных экземпляров объектов.

ResNet



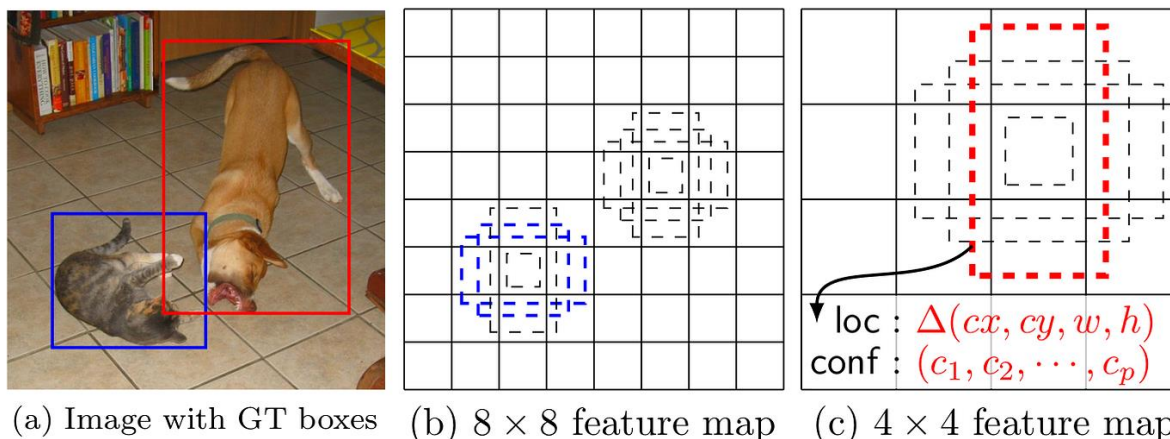
Кульминацией эволюции глубоких сетей в 2015 году стали остаточные сети — ResNet, разработанные командой под руководством Каймина Хэ из Microsoft Research, — которые радикально решали проблему масштабирования архитектур, открыв путь к беспрецедентной глубине без потери стабильности обучения. Сверхглубокие нейронные сети, насчитывающие десятки или сотни слоев, не только рискуют переобучением на ограниченных данных, но и сталкиваются с фундаментальным кризисом оптимизации — так называемым «деградацией точности», когда добавление новых слоев, вместо улучшения, ухудшает общую производительность из-за затухания градиентов во время обратного распространения ошибки, что делает сеть неспособной к эффективному обучению. Решение оказалось элегантным и изобретательным: вместо того чтобы заставлять каждый последующий блок сети вычислять всю функцию преобразования данных с нуля, Хэ предложил учить его лишь «остаточной поправке» — разнице между желаемым выходом и уже имеющимся представлением на входе; для этого вводится «пропускная связь» (skip connection), или остаточный блок, где входной сигнал напрямую «прокидывается» через блок без изменений и складывается с его выходом по формуле $y = F(x) + x$, где $F(x)$ — нелинейное преобразование внутри блока.

Эта структура, вдохновленная идеей, что многие локальные шаги преобразования в очень глубокой модели удобнее мыслить как маленькие корректировки уже хорошего представления. Такой способ записи делает обучение стабильнее: и прямой сигнал, и градиент могут проходить почти без потерь, а глубокие блоки учатся не «тащить на себе весь смысл», а вносить локальные полезные поправки. Поэтому сети на сотни слоёв перестали разваливаться только из-за глубины, позволив гради-

ентам течь свободно через сеть, минимизируя риск их затухания и облегчая оптимизацию даже для моделей с тысячами слоев. В результате ResNet-152 — версия с 152 весовыми слоями — не только избежала дегенерации, но и превзошла более простые аналоги вроде VGG-19 по точности, при этом обладая меньшей вычислительной сложностью: всего около 25 миллиардов операций на изображение против 30 миллиардов у VGG, что сделало ее практичной для развертывания на стандартных GPU. На конкурсе ILSVRC-2015 ансамбль из нескольких ResNet-моделей установил рекорд, снизив ошибку top-5 классификации до фантастических 3,57 процентов на ImageNet — почти вдвое лучше, чем у лидеров 2014 года, — и завоевал первое место во всех треках, включая детекцию объектов. Человеческая точность на ImageNet top-5 обычно оценивается примерно в диапазоне 95–98% в зависимости от подготовки, а наиболее известная историческая цифра, с которой сравнивали сети середины 2010-х, — это 94,9% top-5 точности. Это и объясняет, почему ошибка 3,57% top-5 у ResNet в 2015 году воспринималась как переход через важный психологический рубеж: модель уже сравнялась и даже превзошла ту оценку, которую тогда приняли за «человеческий уровень» на этой конкретной процедуре.

Те же подходы мгновенно интегрировались в другие задачи: на датасете COCO они подтолкнули вперед многообъектную детекцию и семантическую сегментацию, повысив mAP на 5–10 процентов по сравнению с предшественниками, поскольку позволили строить более глубокие извлекатели признаков (бэкбон сети) без потери контекста. Инженерная идея ResNet — «обучение через корректировку» — превратилась в универсальную технологию, применимую не только в классификации, но и в других предсказательных задачах вроде пиксельной сегментации или анализа видео.

SSD



В 2016 году, когда двухэтапные детекторы вроде Faster R-CNN устанавливали новые стандарты точности, но все еще страдали от относительной медлительности из-за разделения на генерацию регионов и классификацию, команда под руководством Лю Вэй — директора лаборатории человеко-машинного взаимодействия и когнитивной инженерии Пекинского университета почты и телекоммуникаций, эксперта в области искусственного интеллекта, представила Single Shot MultiBox Detector (SSD) — радикально упрощенный одностадийный подход, который сильно сдвинул акцент в сторону скорости без серьезных компромиссов в качестве, идеально подходящий для задач реального времени. В отличие от многоступенчатых конвейеров, SSD интегрировала всю логику в единую сверточную сеть, где предсказания рамок и классов объектов генерировались напрямую на множественных уровнях иерархии признаков — от мелких слоев с высокой разрешающей способностью для обнаружения маленьких объектов до глубоких, семантически богатых для крупных, — с использованием предобученного извлекателя признаков вроде VGG-16 или ResNet, адаптированного через дополнительные сверточные слои с якорными окнами разных масштабов и аспектов, чтобы охватывать разнообразие форм. Ключевой инновацией стала «мультибоксовая» стратегия: каждый пиксель на выбранных картах признаков предсказывал не один, а несколько (обычно 4–6) фиксированных боксов с вероятностями классов и смещениями, отфильтрованных через NMS для финального набора детекций; это позволяло сети обрабатывать всю сцену за один проход, достигая 59 FPS на GPU NVIDIA Titan X при mAP 77,2% на PASCAL VOC 2007 — на 10–20% быстрее Faster R-CNN при сопоставимой точности на COCO, где SSD особенно блистал в обнаружении малых объектов благодаря пирамидальной структуре. Для инженеров это означало прорыв в практичности: модель с всего 20–50 миллионами параметров легко разворачивалась на мобильных устройствах или бортовых системах, минимизируя задержки, и быстро стала

основой для приложений вроде автономного вождения или видеонаблюдения.

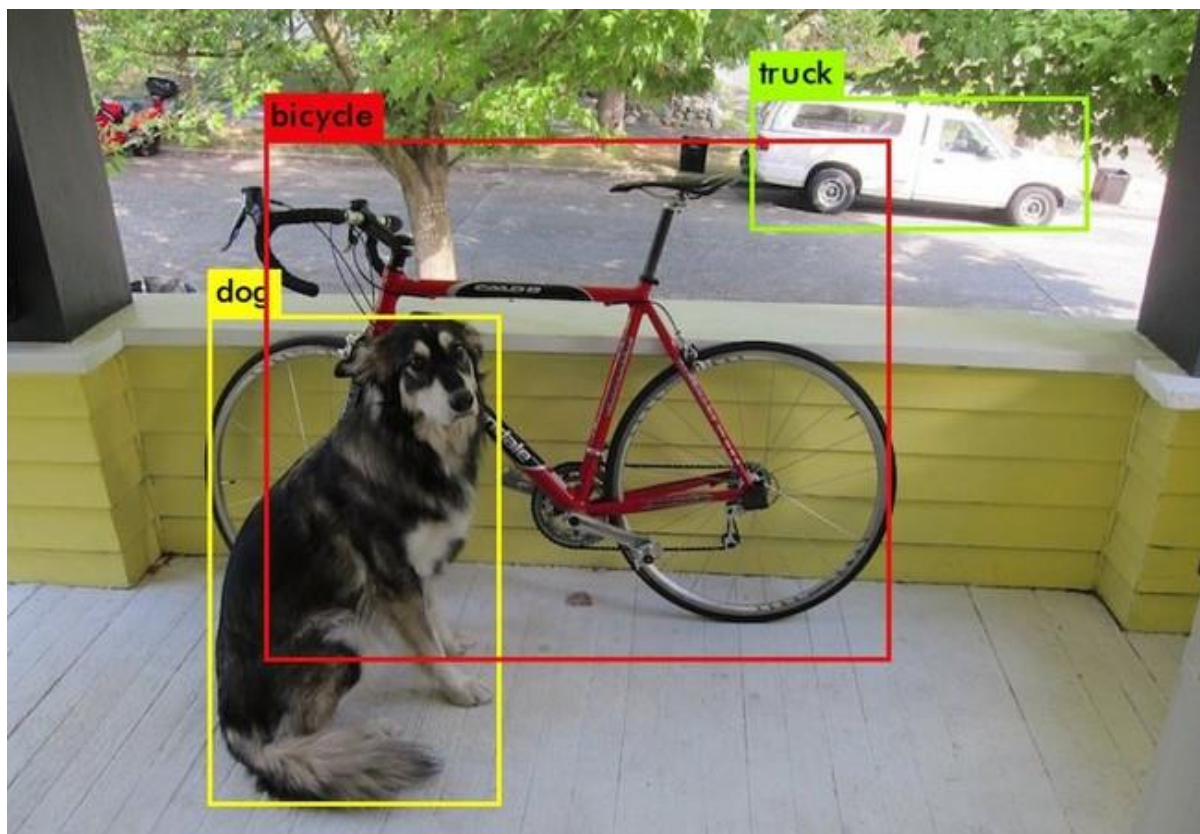
В ходе работы над моим бильярдным проектом, до того, как перейти на YOLO, я долгое время использовал детектор шаров именно этой архитектуры — извлекатель признаков VGGNet с надстройкой SSD (Single Shot MultiBox Detector) размером 300×300 пикселей, для одновременной детекции многочисленных бильярдных шаров. Для дообучения я собрал собственный датасет из 500 цветных изображений типичных игровых ситуаций, каждое с множеством отмеченных и верифицированных шаров. Все инструменты и скрипты для экспорта датасетов пришлось делать самому, включая экспорт в стандартный формат PASCAL VOC. Для любого дообучения важно обеспечить совместимость собственных размеченных файлов с предобученными весами модели в том же формате, что доступны в открытом доступе.

Изначально система работала на CPU и обучалась на Caffe, что позволяло тестировать базовую функциональность, но выявило ключевую проблему: входной размер 300×300 не охватывал всю игровую поверхность стола, поэтому приходилось предварительно через OpenCV нарезать кадр на квадратные плитки по 300×300 пикселей и запускать детекцию минимум четыре раза на изображение, чтобы избежать потери важной информации при простом сжатии. Такой метод сохранял качество за счет отсутствия сильной компрессии, но приводил к накладкам и дубликатам в детекциях объектов, пересекающих стыки плиток, усложняя постобработку. Получив доступ к мощному серверу с GPU для более интенсивных вычислений, я принял решение дообучить расширенную версию VGGNet с размером 512×512 пикселей, где вся значимая область кадра — игровая поверхность — сжимается до максимальной стороны 512 пикселей с сохранением пропорций, а незаполненное пространство заполняется нейтральным фоновым цветом, чтобы избежать искажений. Альтернативный вариант без сохранения пропорций мог бы минимизировать потерю деталей, но потребовал бы подготовки специализированного датасета с соответствующими аугментациями.

Новый подход радикально упростил пайплайн: сеть применялась всего один раз на кадр, что не только устранило артефакты от стыков, но и сэкономило около 200 миллисекунд на обработку с учетом повышенной вычислительной нагрузки модели, обеспечивая плавный анализ. Выбор размера 512×512 обосновывался наличием готовой архитектуры и предобученных весов, а также ее конструктивными отличиями от версии 300×300 : дополнительные сверточные слои и адаптивные модули, оптимизированные для большего масштаба, генерируют более детализированные карты признаков, повышая точность локализации без чрезмерного роста параметров.

SSD предлагает удобный набор параметров для тонкой настройки выходного слоя, позволяя эффективно фильтровать детекции и отсеивать шум, что особенно ценно в задачах машинного зрения с переменными условиями. Среди них — порог NMS, который подавляет дублирующиеся рамки с высоким процентом пересечения, предотвращая множественные срабатывания на одном объекте; веса для классов, регулирующие приоритет распознавания конкретных категорий; параметр `keep_top_k`, ограничивающий количество лучших предсказаний после NMS (обычно 100–200, чтобы избежать перегрузки постобработки); и `confidence_threshold` — ключевой порог уверенности, ниже которого детекции игнорируются (типичные значения 0,25–0,5), что минимизирует ложные положительные результаты без пропуска реальных объектов. Эти опции встроены непосредственно в выходной слой SSD, делая модель гибкой и готовой к интеграции в производственные пайплайны. В контексте фреймворка Caffe и библиотеки OpenCV такое преимущество особенно заметно. SSD позволяет напрямую встраиваться в OpenCV-пайплайны, включая серверы на Node.js через `opencv4nodejs`, без необходимости в отдельном интерпретаторе.

YOLO (You Only Look Once)



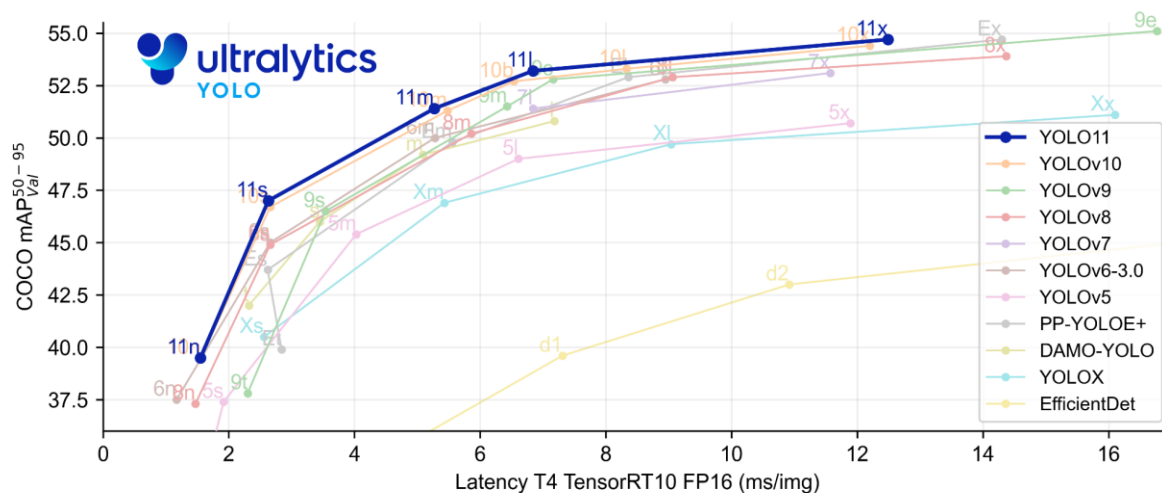
В 2016 году, в начале эволюции одностадийных детекторов вроде SSD, которые мы только что рассмотрели, родилась семья архитектур YOLO — «You Only Look Once», — предложенная Джозефом Редмоном из Университета Вашингтона в соавторстве с коллегами. Вместо многоэтапных конвейеров R-CNN или даже упрощенных одностадийных сетей, YOLO, как и SSD, использовала единую сверточную архитектуру, которая за один прямой проход данных через сеть одновременно предсказывала координаты рамок, вероятности классов объектов и меры уверенности, разделяя изображение на сетку ячеек (обычно 7×7 или 13×13), где каждая ячейка отвечала за несколько предсказаний с якорями разных аспектов, а финальная фильтрация через NMS отсеивала дубликаты. На флагманской тогда видеокарте NVIDIA Titan X базовая YOLOv1 уверенно достигала 45 кадров в секунду при разрешении 448×448 , в то время как упрощенная версия Fast YOLO, с сокращенным числом сверточных слоев и без полносвязных финальных блоков, выжимала впечатляющие 155 FPS — и все это при средней точности (mAP) около 63,4% на PASCAL VOC 2007, что уже сопоставимо с результатами куда более медленных двухэтапных систем вроде Faster R-CNN, но без их вычислительной тяжести в тысячи операций на пиксель. В реальном мире это ощущалось как резкое снятие тормоза: наконец-то стало возможным непрерывно анализировать динамичные сцены — от трафика

на шоссе и конвейерных линий в производстве до уличных камер видеонаблюдения — без задержек в секунды, которые раньше делали нейросетевое машинное зрение практически бесполезным для приложений, требующих мгновенной реакции.

Уже через год, в 2016, Редмон и соавторы выпустили YOLOv2 (известную также как YOLO9000), где радикально улучшили обобщаемость: сеть обучалась совместно на задачах детекции (с локализацией) и классификации (без локализации), используя предобученные веса на ImageNet и технику якорной кластеризации для автоматического подбора оптимальных якорей, что позволило «научить» модель распознавать тысячи категорий, даже если для большинства из них отсутствовали размеченные рамки — в демонстрациях она уверенно идентифицировала более 9000 классов из WordNet-иерархии, сохраняя скорость реального времени на уровне 67 FPS на той же Nvidia Titan X.

В 2018 году вышла YOLOv3, которая закрепила репутацию семейства как эталона скоростной детекции: с входным разрешением 320×320 модель обрабатывала кадр за ~22 миллисекунды при mAP 28,2 на COCO (для всех 80 классов), а при полном 608×608 — достигала 57,9 mAP@0,5 IoU за 51 миллисекунду, опережая по скорости многих именитых конкурентов при аналогичном качестве, благодаря FPN (feature pyramid networks) для многомасштабного предсказания и Darknet-53 как более эффективному извлекателю признаков с остаточными блоками.

На 2026 год компания Ultralytics уверенно развивается, выпуская регулярные обновления. Но прогресс уже идет не столько в глубину (скорость и точность), как в сторону расширения линейки и адаптации. Появляются адаптированные архитектуры для оценки позы людей, трекинг объектов в видеосегментации и прочие.



В целом YOLO и SSD представляют собой две ключевые архитектуры одностадийных детекторов объектов, разработанные в 2015 году

для задач компьютерного зрения, таких как распознавание и локализация объектов в изображениях или видео. Основное различие заключается в подходе к предсказанию: YOLO использует сетку, где изображение делится на ячейки (например, 7×7 или 13×13), и каждая ячейка отвечает за предсказание нескольких рамок с вероятностями классов и уверенности в один проход через сеть, что делает ее полностью сверточной с финальными полносвязными слоями для агрегации. Это обеспечивает высокую скорость, но может снижать точность при обнаружении маленьких или близко расположенных объектов из-за фиксированной сетки. В свою очередь, SSD (Single Shot MultiBox Detector) полагается на многоуровневые карты признаков из извлекателя признаков (например, VGG), генерируя фиксированное множество предсказаний на разных масштабах без полносвязанных слоев — только сверточные, с якорными боксами различных аспектов и размеров для мульти масштабной детекции, что позволяет лучше справляться с объектами разного масштаба и повышает общую точность за счет пирамидальной структуры.

Динамика изменений очень высока, но на текущий момент YOLO опережает SSD по комбинации скорости и точности в сложных сценариях, достигая mAP до 57% на COCO при 20–50 мс на кадр, в то время как базовый SSD уступает в скорости на 10–20% и в точности для динамичных сцен, хотя и сохраняет преимущество в точности локализации. Выбор зависит от приоритетов: YOLO для высокоскоростных приложений (видеонаблюдение, автономное вождение), SSD больше подходит для задач, где критична детализация.

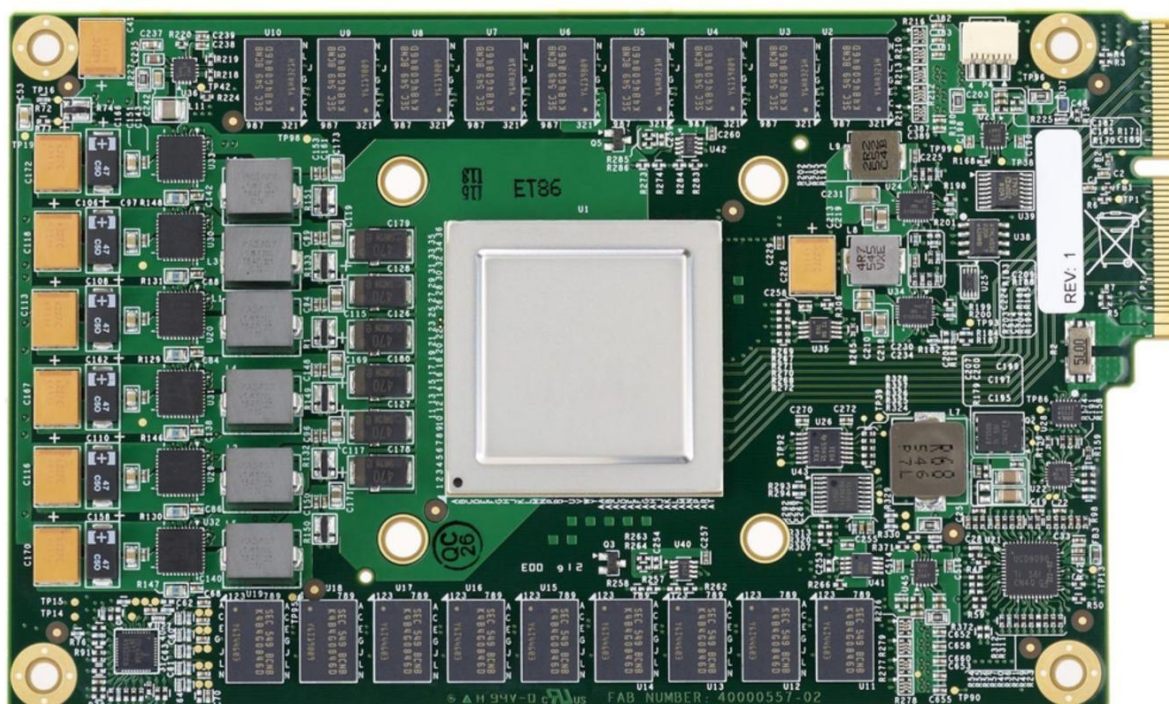
Эволюция SSD после 2016 года была менее динамичной по сравнению с YOLO, но включала несколько значимых улучшений для повышения точности и эффективности. В 2017 году появились варианты вроде DSSD (Deconvolutional SSD), добавившие деконволюционные слои (обратные сверточным) для лучшей детекции маленьких объектов, и FSSD (Feature Fusion SSD), объединяющие признаки из разных уровней для снижения ложных срабатываний. К 2018–2019 годам SSD эволюционировал в RefineDet и RetinaNet (хотя последний ближе к гибриду) с улучшенным извлекателем признаков (ResNet вместо VGG) и модификацией функции потерь — Focal loss. Это модификация обычной кросс-энтропийной функции потерь, придуманная для ситуации, когда в задаче слишком много «лёгких» примеров и слишком мало действительно трудных. В детекции объектов это особенно заметно в одношаговых моделях вроде SSD и RetinaNet: подавляющее большинство окон или ячеек на изображении относится к фону, и сеть очень быстро учится уверенно говорить «здесь ничего нет». Проблема в том, что такие лёгкие отрицательные примеры начинают численно доминировать в обучении и как бы заглушают редкие, но важные сигналы от настоящих объектов.

Идея focal loss в том, чтобы снизить вклад уже хорошо распознанных примеров и, наоборот, усилить вклад трудных и ошибочных. Если сеть и так уверена в правильном ответе, штраф за этот пример искусственно уменьшается. Если же она сомневается или ошибается, штраф остаётся большим. За счёт этого обучение перестаёт тратить основную энергию на миллионы «очевидных пустых мест» и начинает сильнее заниматься тем, что действительно трудно: маленькими объектами, редкими классами, сложным фоном, частичными перекрытиями.

В 2020-х годах SSD интегрировался в легкие модели для портативных устройств (например, MobileSSD), а к 2024–2025 годам появились SSD-подобные архитектуры в комбинации с трансформерами (например, в DETR-вариантах), но доминирование YOLO и многоэтапных моделей вроде Cascade R-CNN ограничило дальнейшее развитие чистого SSD, сделав его нишевым инструментом для специализированных задач с акцентом на точность.

TPU

Практическое применение машинного зрения неизбежно упирается в фундаментальные ограничения аппаратного уровня — обычные CPU и даже GPU часто не справляются с балансом между скоростью, энергоэффективностью и масштабируемостью. В мае 2016 года Google публично раскрыл детали своего первого Tensor Processing Unit (TPU) — специализированного ускорителя, предназначенного исключительно для задач глубокого обучения, который уже с конца 2015 года внутренне обслуживал ключевые сервисы компании, такие как поисковые запросы с визуальным анализом и системы машинного перевода.



Этот чип, построенный на 28-нм техпроцессе, воплощал принцип предельной специализации: в его ядре располагался систолический массив из 256×256 параллельных умножителей-накопителей (MAC-единиц), оптимизированных для 8-битных целочисленных операций с плавающей запятой в низкой точности (bfloat16), что давало в сумме 65 536 параллельных вычислительных блоков и пиковую производительность около 92 триллионов операций в секунду (TOPS) при частоте 700 МГц. В типичных производственных нагрузках — от инференса сверточных сетей до батчевой обработки — TPU обеспечивал 15–30-кратный прирост скорости по сравнению с эквивалентными конфигурациями на CPU или GPU, а по энергоэффективности выигрыш достигал 30–80 раз, что критически важно для масштабируемых облачных сервисов, где каждый ватт мощности напрямую влияет на операционные затраты. Весь фокус

был на «плоском» потоке данных через матричные умножения и программировании через собственный оптимизированный фреймворк TensorFlow, где модель компилируется в статический граф, идеально подходящий для предсказуемого, низколатентного инференса.

TPU не просто доказал жизнеспособность доменно-специфичных ускорителей для нейросетей, но и изменил экономику их развертывания в продакшене: обучение и инференс тяжелых моделей компьютерного зрения — от детекции объектов в видео до генерации синтетических сцен — стали не только быстрее, но и дешевле в эксплуатации, распахнув двери для все более амбициозных архитектур с миллиардами параметров.

TensorFlow



TensorFlow

Запущенный Google в ноябре 2015 года как открытая платформа для машинного обучения, быстро стал краеугольным камнем эры глубоких сетей, открыв инструменты для создания надежных масштабируемых системы, способных перерабатывать миллиарды изображений в реальном времени. Фреймворк, разработанный командой Google Brain под руководством Джеффри Дина и Раджея Монака, изначально фокусировался на статических вычислительных графах — структурах, где модель компилируется заранее для оптимизации производительности, — что сделало его идеальным для промышленных задач, от рекомендационных систем YouTube до анализа трафика в Google Maps.

TensorFlow быстро стал не просто фреймворком для обучения сети, а полным производственным контуром. Вокруг ядра появились `tf.data` для построения входных конвейеров больших данных, TensorBoard для визуализации метрик, графов, эмбеддингов и профилирования, TensorFlow Serving для развёртывания моделей в рабочей среде, а затем и TFX — TensorFlow Extended, уже как платформенный набор для построения и сопровождения ML-конвейеров в продакшене. Экосистема TensorFlow обеспечивает весь жизненный цикл ИИ моделей — от загрузки данных для обучения до обновления версии и мониторинга процессов.

TensorFlow с самых ранних версий внедрил сильную инженерную программу: вычислительный граф нейросети задавался заранее, и это облегчало оптимизацию его исполнения. Но исследователям такой стиль часто мешал — слишком тяжело было быстро отлаживать и менять код. Со временем появилось «немедленное исполнение» (Eager execution), а затем и логика TensorFlow 2, где код можно писать в более естественном императивном стиле, а при необходимости превращать

обратно в граф через `tf.function`. Это принципиальное сближение двух режимов работы: исследовательского и промышленного.

Фреймворк изначально проектировался так, чтобы один и тот же код можно было запускать на CPU, GPU и TPU. В официальных материалах TensorFlow и Google Cloud TPU прямо подчёркивается, что TPU ориентированы на обучение и инференс моделей в крупном масштабе, а TensorFlow предоставляет для этого нативный стек. В тоже время TensorFlow дал отрасли не только серверный стек, но и TensorFlow Lite, а затем появились версии для микроконтроллеров, где модели можно было конвертировать и запускать на маломощных устройствах — в телефонах, камерах, датчиках, встраиваемой электронике. Сейчас эта эпоха переименовывается и развивается дальше, но сам факт остаётся: TensorFlow сыграл заметную роль в том, что нейросетевой инференс вышел в прикладное железо.

В целом экосистема и инфраструктура TensorFlow сделали машинное обучение воспроизводимым и наблюдаемым. До этого многие модели существовали в виде «статьи плюс чей-то внутренний код». TensorBoard, профилировщик, стандартный формат сохранения моделей, типовые учебные примеры и tutoriales, интеграция с Colab — всё это сильно снизило порог входа и ускорило распространение практик, помогло стандартизировать культуру разработки. По данным отчетов Google на 2024 год TensorFlow используется в более чем 100 тысячах проектов по всему миру, от академических исследований до коммерческих решений в здравоохранении и финансах.

PyTorch



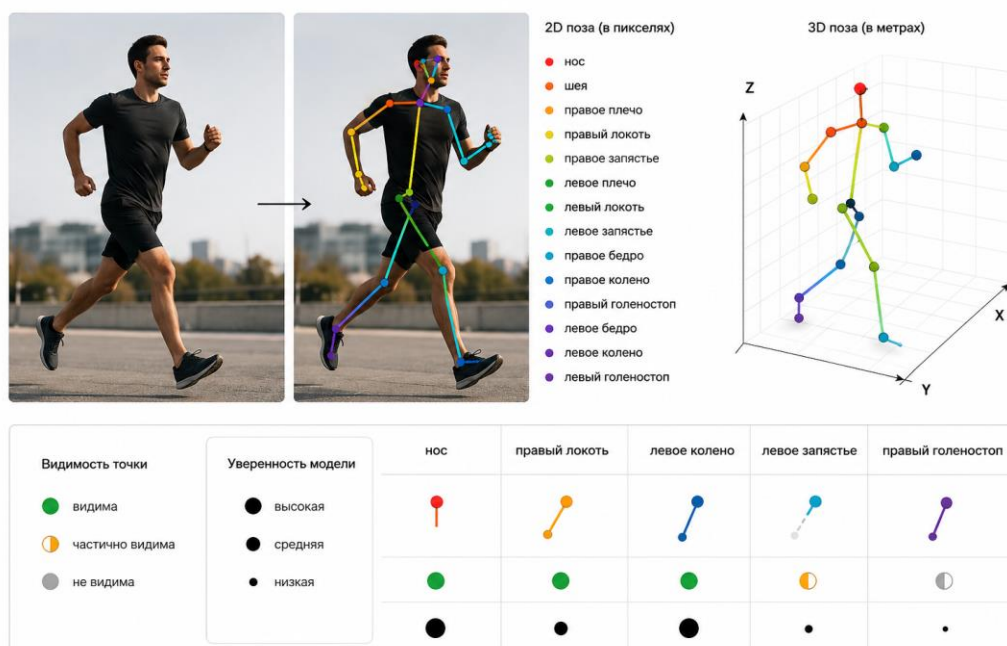
Запущенный в октябре 2016 года командой Facebook AI Research (ныне Meta AI) под руководством Сумита Чинталы, Сэма Гросса и Адама Пашке, с самого начала позиционировался как гибкий инструмент для исследований в области глубокого обучения, вдохновленный предшественником Torch — Lua-ориентированным фреймворком 2002 года. Разработанный на базе Python и C++, PyTorch ввел динамические вычислительные графы — подход, где модель строится и модифицируется в реальном времени, что упрощает отладку и эксперименты, в отличие от ранних статических структур TensorFlow; это сделало его естественным выбором для прототипирования, особенно в машинном зрении, где TorchVision (с 2018 года) предоставляет готовые модели вроде ResNet или Faster R-CNN для задач детекции и сегментации. С выпуском версии 1.0 в декабре 2018 года фреймворк обрел стабильность для продакшена, а PyTorch 2.0 (март 2023 года) добавил `torch.compile` для оптимизации под GPU и TPU, повышая скорость инференса на 30–50% по бенчмаркам MLPerf.

По данным опросов Stack Overflow 2024 года PyTorch доминирует в академических и исследовательских проектах с долей 70%, а его репозиторий на GitHub превысил 80 тысяч звезд, отражая вклад миллионов пользователей; от OpenAI до Google, он стал основой для моделей вроде Stable Diffusion и ViT.

В плане производительности PyTorch выигрывает в гибкости отладки — градиенты и тензоры отслеживаются в реальном времени, — но TensorFlow опережает в оптимизации под аппаратное обеспечение, снижая задержки на 20–30% в распределенных системах по бенчмаркам MLPerf 2025 года. TensorFlow консолидирует позиции в коммерции — его доля в продакшене стабильна на уровне 60% по отчетам O'Reilly AI Survey 2025, с обновлениями вроде TensorFlow 2.17 (май 2025), усиливающими совместимость с JAX для научных вычислений.

Распознавание позы

В задачах распознавания движения всё начинается с позы, мы просим модель не просто «увидеть человека», а разложить его на систему ориентиров — ключевые точки, которые вместе образуют скелет: плечи и локти, бедра и колени, кисти, иногда — черты лица. Это уже не «есть/нет», как в классификации, и не «где объект», как в детекции, а более тонкое описание: набор координат, из которых восстанавливается положение и конфигурация тела. В плоском, двумерном случае координаты выражаются в пикселях кадра (часто еще и приводятся к единому масштабу, чтобы можно было сравнивать людей разного размера). В трехмерном — это точки в реальном пространстве, со своими осями и единицами измерения. В обоих случаях у каждой точки есть признак «видимости»: понятный флаг, который говорит модели и человеку, что локоть закрыт рюкзаком или выходит за край кадра, и уверенность — насколько она сама убеждена в поставленном пикселе.



Разметка для позы — это, по сути, интерактивная геометрия. Аннотатор ставит маркеры на суставы и другие опорные места, двигая их по изображению до совпадения с воображаемым скелетом. На спортивных кадрах это видно быстро и разборчиво; на производстве, где рабочий в форме, с перчатками и инструментами, — заметно труднее: рука оказывается «слита» с фоном, а локоть скрыт. В 3D все ещё сложнее: нужны либо лабораторные данные с маркерами и десятком камер, либо аккуратная настройка многокамерной системы на реальном объекте с точной калибровкой. Иначе говоря, необходимо знать, как соотносятся мировые координаты с каждым изображением, — без этого невозможна

честная реконструкция глубины. Из школьной оптики достаточно помнить одну вещь: плоский кадр — это проекционная тень трехмерного мира, и чтобы превратить ее обратно в пространство, нужно либо несколько «теней» с разных сторон, либо сильные предположения о том, как именно может сгибаться человеческое тело.

Где это применяют? В спорте — от анализа техники бега до оценки углов в суставе при подаче в теннисе. В фитнесе — обратная связь по повторениям и амплитудам, когда приложение в телефоне понимает, что присед закончен, а спина остается прямой. В безопасности и эргономике на производстве из «скелета» вычисляют простые индексы риска: сколько секунд работник проводит с поднятой рукой, как часто изгибается шея, на каких углах колени в момент подъема груза. Для дополненной реальности и интерфейсов жестов точность на уровне кистей и пальцев — это очень функционально: ошибись на сантиметр — и курсор улетит мимо кнопки.

У распознавания позы есть естественные подзадачи. Иногда нас интересует один-единственный герой, выбранный в первом кадре: нужно не потерять его среди остальных, когда он поворачивается боком или уходит на мгновение за препятствие. Это похоже на трекинг одного объекта, только вместо рамки мы держим «скелет». Гораздо чаще нужно разобрать всю сцену сразу: на баскетбольной площадке десять человек, и у каждого — свои колени и плечи. Здесь удобны два разных маршрута. Первый — сверху вниз: сначала находим каждого человека рамкой, потом внутри рамки строим «скелет». Этот путь прост и часто дает устойчивые результаты, потому что видит каждого человека крупнее и чище. Второй — снизу вверх: модель сразу «ищет» все возможные ключевые точки на кадре, а затем связывает их в отдельных людей, опираясь на геометрию. Такой подход лучше переживает толпы и перекрытия, но требует аккуратного «сведения» точек в правильные скелеты. В обоих случаях современным моделям помогает способность «держат высокое разрешение» внутри себя и смотреть на позу не только локально, но и с учетом положения других частей тела (High-Resolution Nets, трансформерные позовые модели — примеры таких идей).

Как понять, хорошо ли мы определяем позу? В двумерном мире полезно мыслить не абсолютными пикселями, а относительными расстояниями: одна и та же ошибка в пять пикселей на лице крупным планом и на человеке вдалеке — это разная ошибка. Поэтому классическая мера считает точку «правильной», если она попала в окружность разумного радиуса вокруг истинного положения, и этот радиус растет с размером человека на картинке. Дальше считаем долю «правильных» по всем точкам и всем людям — получаем понятный процент. На больших наборах данных используют более «тонкую» версию той же идеи: у каждой

точки свой «характерный» размер погрешности, и попадания взвешиваются с учетом видимости и сложности. Для 3D логика другая: мы можем измерять среднее расстояние в миллиметрах между предсказанным суставом и эталонным, усредняя по всем суставам и кадрам. Получается честная метрика «сколько в среднем промахиваемся», и это число легко интерпретировать. Есть нюансы — но смысл один: чем меньше средняя ошибка, тем лучше.

Типичные проблемы оценки позы обусловлены положением человека. Первая — окклюзии: часть тела закрыта другой частью, предметом, человеком. Модель начинает «догадываться» по геометрии, и эти догадки не всегда верны. Вторая — необычные позы, далекие от «классической ходьбы» из учебных примеров: акробатические элементы, прыжки с разворотом, сидячие позы с перекрещенными ногами. Если подобного не было в обучении, сеть теряется. Третья — масштаб и ракурс: ребенок и баскетболист одного роста на изображении занимают разную долю кадра, а человек, снятый снизу, выглядит иначе, чем тот же человек, снятый сверху. Четвертая — одежда и инвентарь: свободная куртка прячет локоть, ракетка закрывает кисть, рюкзак смешивает очертания плеч. Пятая — банальные «лево-право»: зеркальные отражения и повороты иногда приводят к тому, что локоть «перепрыгивает» с одной руки на другую, особенно в толпе.

Инструменты борьбы с этими ошибками в целом предсказуемы. Самое мощное — данные. Если мы можем «поиграть в режиссера» и снять разнообразные сцены — с перекрытиями, ракурсами, в разных костюмах, — модель быстро научится опираться не на случайные подсказки, а на устойчивую геометрию тела. Когда такой роскоши нет, помогает синтетика: сегодня вполне реально генерировать реалистичные кадровые наборы людей, когда вручную заданы позы, освещение и одежда, и использовать их для «прививки» модели к редким ситуациям. В видео помогает время: поза — не случайное перемещение, а плавное движение, и даже простое сглаживание траекторий ключевых точек убирает дрожание и случайные скачки, не съедая быстрые, но реальные движения. Если система многокамерная, включается геометрия: при известной калибровке пары кадров из разных мест достаточно, чтобы восстановить 3D-точку по ее двум проекциям. Это та самая школьная «треугольная засечка», только в цифровом виде: чем точнее вы знаете, где стоят камеры и куда они смотрят, тем надежнее глубина.

Два примера покажут, как эти принципы работают на практике. Представьте мобильное приложение для домашнего фитнеса. Телефон стоит на штативе, человек делает выпады. Задача — понять, когда повтор завершен, и подсказать про угол в колене. Камера одна, освещение

и фон случайные. Модель сначала находит человека, затем строит «скелет» в 2D и измеряет угол между бедром и голенью. Чтобы не раздражать пользователя «дергаными» числами, углы сглаживают по времени, а порог завершения повторения делают зависимым от амплитуды конкретного человека — высокому и гибкому он один, невысокому и начинающему — другой. Отдельно проверяют устойчивость к одежде: тренируем на коротких шортах и длинных спортивных штанах, чтобы колено не «терялось» под тканью. Для редких поз — выпад назад с поворотом корпуса — при обучении добавляют синтетические кадры, сгенерированные в движке, и небольшую дозу реальных примеров.

Другой пример — безопасность труда на складе. Камеры установлены сверху, задача — фиксировать опасные наклоны спины при подъеме коробок. Здесь важнее не сантиметровая точность кистей, а надежность определения «спина согнута/прямая». Позу можно хранить в «скелете» из нескольких крупных звеньев: линия плеч, линия бедер, ось спины. Угол между плечами и бедрами превращается в простой индикатор риска. Чтобы система не рушилась при движении камеры (легкий «дрейф» креплений) и в тени высокого стеллажа, настраивают порог предупреждения с небольшим запасом, а в модель добавляют «память» на несколько секунд, чтобы единичные провалы по видимости не рвали события. Ложные срабатывания дороже пропусков? Тогда сигнал выводят сначала как «мягкую подсказку» на панель бригадира, а «жесткую тревогу» — только при устойчивом нарушении дольше заданного времени.

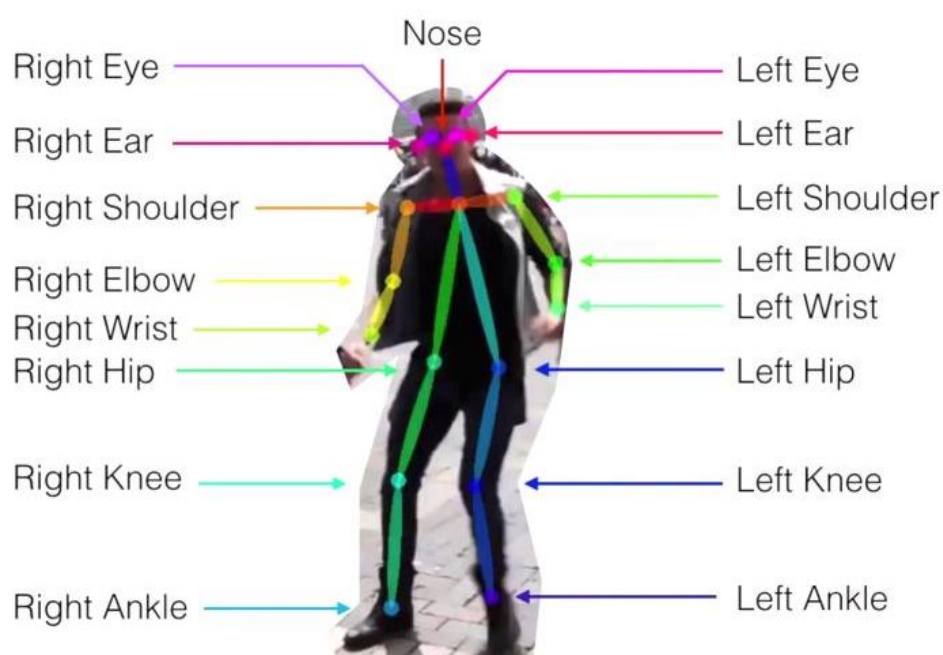
Трехмерная поза — отдельная история и головная боль. С одной стороны, это настоящая «кинематика»: можно измерять углы в реальном пространстве, считать скорость звеньев и оценивать нагрузки. С другой — неоднозначность глубины из одной камеры: протяни руку вперед или подними в сторону — на плоском кадре это выглядит похоже, а в 3D — разные конфигурации. Поэтому в коммерческой практике либо ставят две-три камеры и честно «триангулируют» точки, либо используют сильные модели тела, которые задают не просто набор шарниров, а «приемлемое семейство поз» и подбирают ту, что лучше всего объясняет картинку. Первый путь дает физически точные параметры в миллиметрах при хорошей калибровке; второй — работает из одной камеры, но требует аккуратной настройки и «проверки здравого смысла» на редких позах.

И, наконец, важно помнить, что распознавание позы — это не только математика, но и ответственность. Мы работаем с людьми, часто — в рабочих процессах и спорте, где решения системы, принятые на основе распознанного «скелета», влияют на нагрузку и безопасность. Это

значит, что система должна честно показывать свою уверенность, давать оператору возможность перепроверить и исправить, и держать в себе простые ограничения, которые предотвращают абсурдные положения (локоть не может внезапно оказаться на месте колена). С инженерной точки зрения это звучит прозаично: немного проверки данных, несколько строк сглаживания, аккуратная калибровка и благоговение перед реальностью. С пользовательской — это разница между игрушкой и инструментом, которому доверяют.

OpenPose

В 2017 году научное сообщество и разработчики компьютерного зрения познакомились с OpenPose — инновационной открытой системой, созданной в Карнеги-Меллонском университете под руководством Ясера Шейха и его команды, включая Чжэ Цао, Гинеса Идальго и других. Эта платформа обеспечила оценку позы в режиме реального времени, одновременно выявляя «скелеты» — ключевые точки суставов и конечностей — у нескольких людей: от основных частей тела (плечи, локти, колени) до детализированных элементов рук, лица и, в последующих обновлениях, стоп (что особенно ценно для задач захвата полного движения, включая создание цифровых аватаров в виртуальной реальности).



Ключевым техническим достижением стали поля ассоциации частей (Part Affinity Fields, PAFs) — векторные карты, генерируемые сверточной нейронной сетью, которые не просто локализуют отдельные точки, такие как локоть или запястье, но и кодируют их пространственные связи и принадлежность к конкретному человеку, даже в плотной толпе или при перекрытиях фигур; это позволило алгоритму решать проблему многопользовательской ассоциации с точностью до 60–70% на бенчмарках вроде COCO Keypoints, минимизируя ошибки в сложных сценариях. Редкостью для академических разработок стало одновременное опубликование подробной статьи в IEEE TPAMI и открытого исходного кода на GitHub под лицензией BSD, что сделало переход от теории к практике мгновенным: инженеры могли скачать репозиторий, собрать исполняемый файл за часы и интегрировать систему в приложения — от фитнес-трекеров для анализа упражнений до аналитики в спорте или интерактивных инсталляций в розничной торговле.

OpenPose быстро эволюционировал в эталонный инструмент для оценки позы, аналогичный роли OpenCV в базовых алгоритмах обработки изображений: он открыл доступ к сложной задаче, позволив сотням разработчиков, не имеющих глубоких знаний, эффективно решать реальные проблемы в области оценки движений, робототехники и анимации, и к 2018 году набрал миллионы загрузок, вдохновляя на десятки форков и расширений.

Интересно отметить, что в 2017–2018 годах команды из Яндекса и Сколтеха (Сколковский институт науки и технологий) тоже отправляли свои решения в подзадачи вроде оценки позы и генерации описаний изображений (captioning), достигая mAP около 40–45%, что было конкурентно, но уступало лидерам вроде китайского SenseTime или американского Facebook AI.

В 2023 году я работал над небольшим стендовым проектом — системой анализа движений, где мы сознательно сделали ставку не на лабораторную точность, а на доступность и воспроизводимость в реальных условиях. Это было мобильное приложение, и принципиальное решение заключалось в том, что вся основная логика выполнялась прямо на устройстве — без отправки видео в облако. Камера телефона становилась единственным сенсором, а вычисления — частью пользовательского опыта, а не скрытой серверной инфраструктуры.

В основе лежал пайплайн из двух частей. Сначала модель уровня YOLO решала задачу быстрой детекции и стабилизации человека в кадре, отсекая лишние области и обеспечивая устойчивость при движении, смене фона и освещении. Поверх этого работал блок оценки позы на базе OpenPose, который извлекал ключевые точки тела и превращал видеопоток в последовательность скелетных конфигураций. Важным было не просто «найти позу», а сделать это с достаточной частотой и стабильностью, чтобы можно было анализировать динамику — ускорения, амплитуды, ритм.

Конкретная задача, с которой мы начали, выглядела на первый взгляд простой: подсчёт подпрыгиваний со взмахами рук за единицу времени. Но за этим стояло несколько нетривиальных решений. Нужно было устойчиво отделить реальное движение от шумов — дрожания камеры, частичных перекрытий, ошибок детекции. Мы вводили геометрические и временные критерии: отслеживали вертикальное смещение центра массы, синхронизацию движений рук и ног, характерные фазы цикла «приседание — отталкивание — полёт — приземление». Каждый эпизод проверялся не только по одному признаку, а по согласованности нескольких сигналов, чтобы избежать ложных срабатываний.

Дальше начиналась более интересная часть — переход от счётчика к интерпретации. Мы стандартизировали показатели: вводили нормировки по росту, длине конечностей, частоте кадров, условиям съёмки. Из сырых измерений — количества повторений, амплитуды, скорости — формировались устойчивые метрики, которые можно было сравнивать между пользователями и между сессиями одного и того же ребёнка. Это позволило выйти на уровень методологии: не просто «сколько раз подпрыгнул», а как именно выполняется движение, насколько оно координировано, ритмично, симметрично.

На этой базе мы начали выстраивать систему оценки функциональных навыков. Идея была в том, чтобы через простые, но хорошо измеряемые упражнения получить косвенную оценку общей двигательной подготовки: координации, взрывной силы, способности к ритмическому повторению движения. Важный момент — это не медицинская диагностика и не жёсткая классификация, а именно ****ранняя оценка predisposedности и уровня развития****. Система позволяла увидеть, где ребёнок двигается уверенно и стабильно, а где возникают характерные «провалы» — рассинхронизация, потеря амплитуды, нестабильный ритм.

С инженерной точки зрения этот проект для меня был важен тем, что он подтвердил простую вещь: даже относительно лёгкие модели, работающие на мобильном устройстве, дают практическую ценность, если правильно поставить задачу и аккуратно собрать признаки. Здесь не требовалась идеальная реконструкция позы или сложная биомеханическая модель. Достаточно было устойчиво извлекать ключевые точки, правильно интерпретировать их во времени и построить вокруг этого внятную систему метрик. В результате мы получили инструмент, который можно использовать вне лаборатории — в школе, спортивной секции или дома — и который при этом даёт воспроизводимые и сопоставимые результаты.

EfficientNet

Ближе к 2020 году внимание исследователей в области компьютерного зрения вновь сосредоточилось на вопросах эффективности и оптимизации, где растущая сложность моделей сталкивалась с ограничениями аппаратных ресурсов. В 2019 году Мингсинг Тан (Mingxing Tan) и Куок Ле (Quoc V. Le) из Google Brain представили EfficientNet — архитектуру, основанную на относительно простой, но фундаментально обоснованной идее: вместо изолированного масштабирования глубины, ширины или разрешения входных изображений следует применять согласованное увеличение всех трех измерений с использованием единого компаунд-коэффициента (compound scaling coefficient), определяемого эмпирически для достижения оптимального баланса производительности и затрат. Базовая конфигурация EfficientNet-B0 была получена с помощью автоматизированного поиска архитектур (Neural Architecture Search, NAS), который выявил эффективную комбинацию стандартных сверточных блоков с инвертированными остаточными связями (inverted residuals) и squeeze-and-excitation модулями для динамического внимания к каналам.

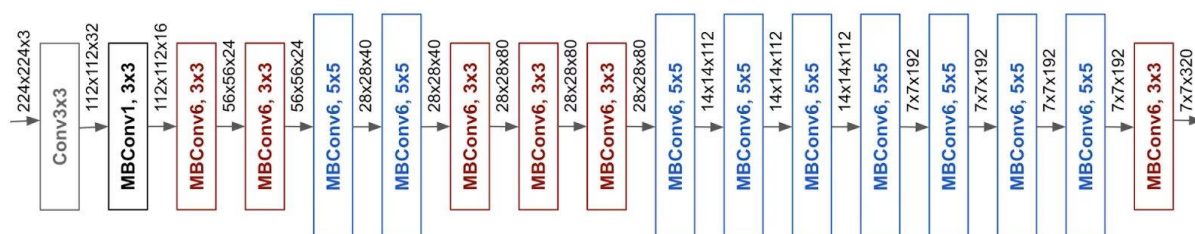
Модули squeeze-and-excitation по-русски удобнее всего называть модулями сжатия и возбуждения или, чуть свободнее, модулями перенастройки каналов. Их предложили как способ заставить сверточную сеть не просто извлекать признаки, а ещё и понимать, какие каналы признаков в данный момент действительно важны, а какие стоит приглушить. Авторы прямо формулировали цель как явное моделирование зависимостей между каналами и адаптивную перенастройку откликов по каналам при небольшой вычислительной цене.

Если объяснить идею простыми словами, то обычный сверточный блок после обработки изображения выдаёт пачку карт признаков — десятки или сотни каналов. Один канал может реагировать на границы, другой — на текстуру, третий — на контрастные пятна, четвёртый — на более сложную форму. Но сама сеть без дополнительного механизма не всегда явно решает, какие из этих каналов сейчас особенно полезны для конкретного изображения. Модуль сжатия и возбуждения вставляется после сверточного блока и делает короткую внутреннюю проверку: сначала он сжимает пространственную информацию каждого канала до одного числа, то есть получает очень компактное резюме по каналу; затем по этим числам вычисляет, какие каналы надо усилить, а какие ослабить, и умножает исходные карты признаков на соответствующие коэффициенты. По сути это форма внимания к каналам, а не к пикселям.

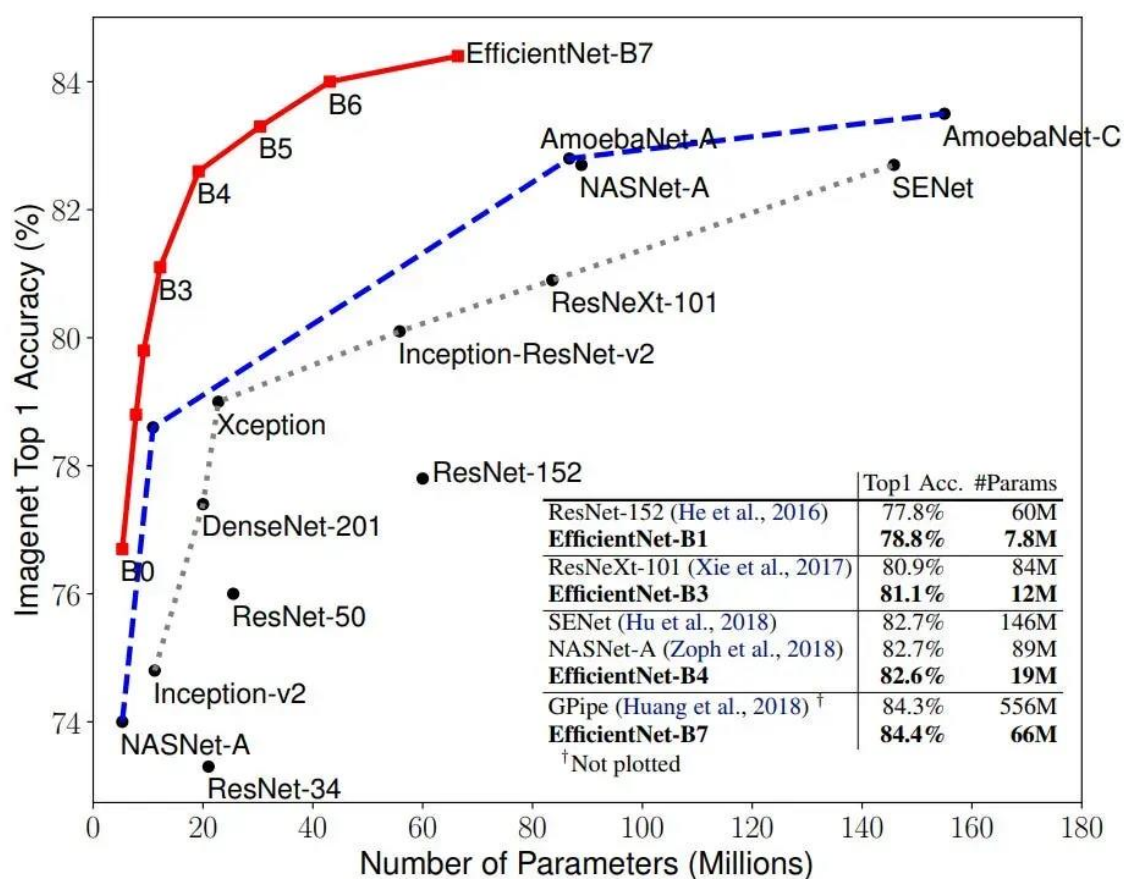
Слово squeeze здесь означает именно «сжатие»: из большой двумерной карты признака берётся её глобальное усреднённое описание. Сеть как будто перестаёт смотреть на точное расположение откликов и

спрашивает только: насколько вообще этот канал активен на всём изображении. Слово *excitation* — «возбуждение» — обозначает следующий шаг: маленькая вспомогательная сеть по этому краткому описанию решает, как перенастроить веса каналов. Если канал сейчас важен, его коэффициент становится ближе к единице или выше; если менее важен — он приглушается. Это не создаёт новых признаков, а переоценивает уже найденные, подстраивая сеть под содержимое конкретного входа.

Почему это оказалось полезно именно для EfficientNet? Потому что EfficientNet строится вокруг идеи максимально экономной сети, где каждая операция должна приносить заметную пользу. Модули сжатия и возбуждения хорошо подходят под эту философию: они дают сети способ тонко перераспределять внимание между каналами при относительно малой цене по параметрам и вычислениям. В архитектуре EfficientNet базовый блок — это уже достаточно эффективная схема с инвертированными остаточными связями и разделёнными по глубине свёртками, а модуль сжатия и возбуждения добавляет к этому ещё один слой адаптивности: сеть не просто считает признаки, а сразу же решает, какие из них в данном контексте усилить. В самой статье про EfficientNet это прямо видно по схеме базового блока: SE-модуль встроен как часть стандартного строительного элемента B0.

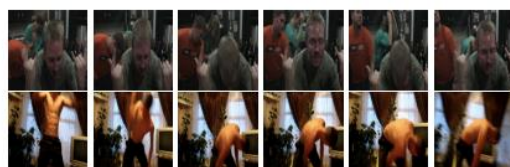


На этой основе авторы создали семейство моделей от B0 до B7, где каждый последующий вариант увеличивал коэффициент масштабирования в 1,2 раза по глубине, 1,1 раза по ширине и 1,15 раза по разрешению. Обратите внимание, насколько изоощренно подошли к делу создатели проекта — применили механизм автоматизированного поиска архитектур. Алгоритмы уже сами ищут пути для усовершенствования самих себя, например адаптируют под конкретные устройства, или под определенные типы объектов, для повышения общей производительности.



Найденный новый подход продемонстрировал выдающуюся эффективность: флагманская модель EfficientNet-B7 достигла 84,3% точности top-1 на датасете ImageNet при всего 66 миллионах параметров и 37 миллиардах операций с плавающей запятой (FLOPs), что в 8,4 раза меньше параметров и в 6,1 раза меньше вычислительных затрат по сравнению с ведущими сверточными аналогами того времени, такими как GPipe или AmoebaNet, при сопоставимой или лучшей точности. Для индустрии это открытие имело стратегическое значение: оно подчеркнуло возможность рационального распределения вычислительного бюджета, позволяя интегрировать высокопроизводительные сети в ресурсоограниченные устройства — от смартфонов и камер умного дома до носимых гаджетов, — где традиционные гигантские модели были непрактичны. EfficientNet не только установил новый эталон для сверточных сетей, но и повлиял на последующие разработки, включая гибридные подходы с трансформерами и оптимизированные версии для периферийных вычислений, подтвердив, что целенаправленная оптимизация может существенно снизить барьеры для широкого внедрения компьютерного зрения в повседневные приложения.

Kinetics-400 и 3D ConvNet



(a) headbanging



(c) shaking hands



(e) robot dancing



(b) stretching leg



(d) tickling



(f) salsa dancing

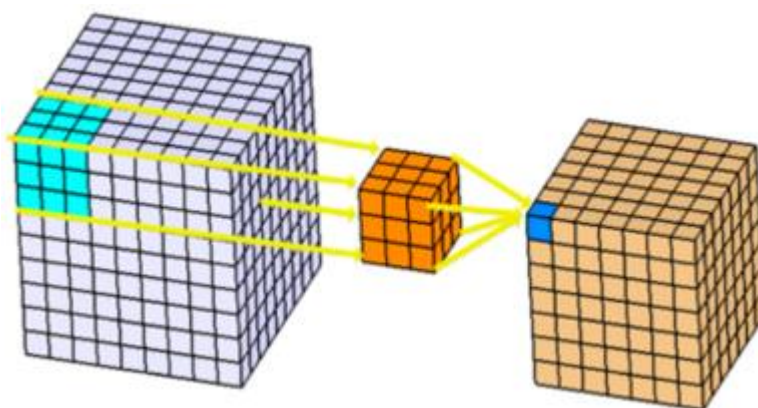
От распознавания образов на статичных изображениях плавно переходим к описанию действий в видео. В 2017 году благодаря усилиям команды из Google DeepMind под руководством Жоао Каррейры и Эндрю Зиссермана — кого я ранее уже упоминал в работах по классификации изображений в VGG, увидел свет Kinetics-400, один из фундаментальных бенчмарков для распознавания действий в видео. Этот датасет, насчитывающий около 300 тысяч коротких клипов длительностью по 10 секунд каждый, был собран из видео на YouTube и охватывает 400 разнообразных классов человеческих действий — от повседневных, как "наливание кофе" или "разговор по телефону", до динамичных, вроде "прыжков через скакалку" или "танцев под хип-хоп". Каждое видео аннотировано с учетом временных меток, чтобы облегчить обучение моделей на точных отрезках, а классы сгруппированы в 34 категории по типам (спорт, быт, эмоции), что обеспечивает баланс и репрезентативность реального мира. Kinetics-400 стал катализатором для эволюции видеоанализа: до его появления модели вроде двухпоточных CNN тестировались на скромных наборах вроде UCF-101, но этот датасет поднял планку, требуя от алгоритмов — от 3D-сверток SlowFast до трансформеров TimeSformer — способности улавливать не только статические образы, но и временную динамику. Об этих алгоритмах мы с вами еще поговорим далее.

Заметный вклад именно в машинное зрение внёс Жоао Каррейра. Его ранние работы тоже связаны с Оксфордом и группой Visual Geometry

Group, где он занимался задачами распознавания действий и анализа видео. Позже он перешёл в DeepMind, где продолжил работать уже в более широком контексте обучения представлений. В его профессиональной траектории хорошо видно, как исследователь из «классического» компьютерного зрения постепенно оказывается в центре современной ИИ-инфраструктуры.

Интересен и его стиль мышления. В его работах нет стремления полностью перепридумать архитектуру. Скорее наоборот — он ищет способ взять уже устойчивое решение и расширить его область применимости. Это довольно редкое качество: не изобретать новую модель, а понять, где старая перестаёт работать, и аккуратно дотянуть её до следующего уровня сложности. В случае Каррейры этим «следующим уровнем» стало видео.

Основное направление, с которым его обычно связывают, — это развитие трёхмерных свёрточных сетей. Как раз на датасете Kinetics родилась его работа «Куда движется распознавание действий? Новая модель и датасет Kinetics». Именно в ней была представлена модель I3D (Inflated 3D ConvNet) — тот самый подход с «раздуванием» 2D-свёрток в 3D. Если обычная свёртка работает по двум измерениям изображения — ширине и высоте, — то здесь добавляется третья ось: время. Вместо того чтобы анализировать каждый кадр отдельно, сеть начинает видеть небольшой фрагмент видео как единый объём данных. Это означает, что фильтр реагирует не только на форму, но и на изменение формы во времени — на движение.



Ключевая идея, которую Каррейра предложил вместе с коллегами, заключалась в том, чтобы не строить такие сети с нуля, а «раздуть» уже обученные двумерные модели в трёхмерные. То есть взять фильтры, которые хорошо работают на изображениях, и распространить их на временную ось. Это позволило использовать накопленные знания из больших датасетов изображений и перенести их в видео-задачи. Такой ход оказался очень практичным: сеть не начинала обучение «с чистого листа», а сразу получала разумное начальное представление.

С технологической точки зрения трёхмерные свёртки решают важную проблему: они позволяют объединить в одном механизме пространственные и временные признаки. До этого часто приходилось комбинировать отдельные компоненты — например, свёрточную сеть для кадра и отдельную модель для движения. Здесь же движение становится частью того же самого представления. Это делает модель более цельной, но одновременно усложняет её: резко растёт объём вычислений, требования к памяти и данным.

Поэтому развитие этой линии шло в двух направлениях. С одной стороны, искали более эффективные варианты трёхмерных свёрток, чтобы не утонуть в вычислительной сложности. С другой — учились правильно подбирать временной масштаб: сколько кадров нужно анализировать одновременно, чтобы уловить действие, но не перегрузить модель. Эти компромиссы до сих пор остаются актуальными, даже в эпоху трансформеров.

Когда я впервые посмотрел на идею трёхмерных свёрток, она показалась естественной для бильярда. У меня уже было устойчивое представление сцены: игровое поле в нормализованном виде около 1280×640 пикселей. Добавляя третью ось — время, я получал объём данных: примерно 10 секунд по 30 кадров в секунду, то есть порядка 300 «срезов» по глубине. В таком виде поведение шаров — это уже не набор отдельных кадров, а непрерывная траектория в пространстве-времени. Логично было попробовать обрабатывать это как единый 3D-объём.

Я разбивал эту структуру на локальные трёхмерные блоки и обучал сеть находить в них закономерности. Архитектуры подбирали вручную на Python с использованием Keras — без сложных автоматизированных поисков, просто через серию экспериментов. Довольно быстро стало понятно, что исходный объём слишком велик: память, скорость, стабильность обучения — всё начинало работать на пределе. Тогда я сделал агрессивное упрощение: по сути «укрупнил» сетку в 10 раз. Если в кубе $10 \times 10 \times 10$ появлялись точки движения, они объединялись в одну ячейку. На временном масштабе около трети секунды это ещё сохраняло правдоподобную динамику шаров.

Именно здесь проявилась ключевая сложность — согласование времени и пространства. Если ячейка слишком крупная, теряется точность траектории; если слишком мелкая — система становится вычислительно неподъёмной и начинает шуметь. Нужно было найти такой масштаб, при котором движение шаров остаётся различимым, но представление остаётся компактным. Это оказалось гораздо менее тривиальной задачей, чем казалось вначале: в бильярде важны сантиметры и доли секунды, а 3D-свёртка склонна «размывать» эти различия.

Во-первых, я хотел обучить модель валидировать сам факт удара и определять результат — промах или попадание. Во-вторых, я пробовал использовать этот же 3D-объём для трекинга и переидентификации шаров: идея была в том, что траектория в пространстве-времени сама по себе является сильным идентификатором. В теории это выглядело убедительно: каждый шар оставляет характерный «след» в объёме.

На практике этот подход не дал достаточной точности. Для задачи валидации удара модель уступала более простым и специализированным решениям, которые опирались на явные признаки и физическую интерпретацию. Для трекинга ситуация была ещё жёстче: 3D-свёртки плохо справлялись с необходимостью точного соответствия объектов, особенно при пересечениях траекторий и окклюзиях. Слишком много информации терялось на этапе укрупнения и агрегации.

В итоге я от этого направления отказался. Но сам эксперимент оказался полезным. В бильярде важна не только динамика, но и точная геометрия, дискретные события, физические ограничения. 3D-свёртка пытается усреднить всё это в одном представлении, и именно это усреднение здесь оказалось лишним. Для меня это был важный этап работы: попытки применить разные подходы к одной и той же задаче — единственный способ сравнить их эффективность.

Трансформеры

Трансформеры были предложены в 2017 году в работе «Attention Is All You Need» как альтернатива рекуррентным моделям последовательностей. Авторы, среди которых Ашиш Васвани, Ноам Шазир, Ники Пармар, Якоб Ушкорейт, Ллион Джонс, Айдан Гомес, Лукаш Кайзер и Илья Полосухин, решали тогда узкую, но важную задачу — машинный перевод. До этого почти все модели строились на рекуррентных сетях: текст обрабатывался последовательно, слово за словом, с передачей состояния. Это было медленно, плохо масштабировалось и с трудом удерживало длинный контекст. В новой работе они предложили отказаться от рекуррентной структуры и построить модель, которая сразу смотрит на всю последовательность и учится перестраивать её представление через механизм внимания. Отсюда и появилось название «трансформер», что буквально означает: модель, которая трансформирует (преобразует) одну последовательность представлений в другую.

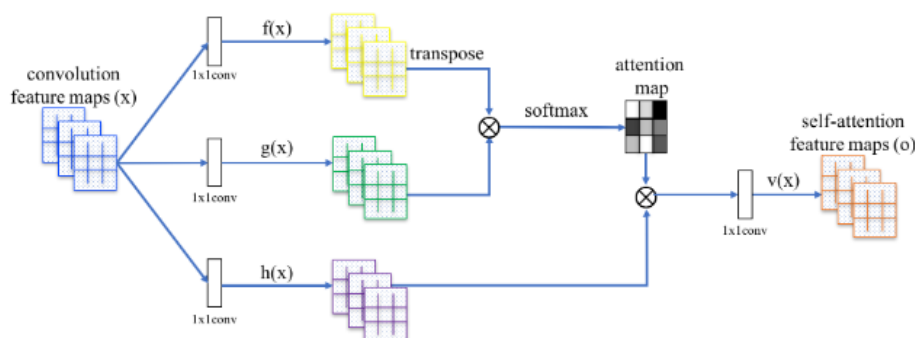
Но важно, что именно преобразуется. Не сами слова, а их внутреннее представление. На каждом слое модель берёт набор векторов (слов или токенов), вычисляет между ними взаимные связи через self-attention и строит новую, более информативную версию этих векторов. То есть она не «обрабатывает поток», а многократно переписывает описание всей последовательности, делая его всё более согласованным.

Исторически название закрепилось по аналогии с более ранними архитектурами вида encoder–decoder, где модель «преобразует» вход в выход. Но здесь преобразование стало центральной идеей: не запоминание состояния, как в рекуррентных сетях, и не локальная фильтрация, как в свёртках, а именно глобальное преобразование структуры данных через внимание.

Интересно, что это название оказалось неожиданно удачным. Оно не привязано ни к тексту, ни к речи, ни к изображениям. В результате архитектура быстро вышла за пределы перевода и стала универсальной: трансформеры применяются в обработке текста, изображений, видео, аудио — везде, где есть структура, которую можно преобразовывать.

Ключевая инновация — нейросетевой слой-алгоритм self-attention, математически имитирующий механизм внимания. В нём, в отличие от сверточных слоев, которые математически опираются на локальные, иерархические фильтры, реализуется глобальная ассоциация через скалярное произведение запросов, ключей и значений (Q, K, V), где выходной слой (softmax) нормализует релевантность пар элементов в последовательности, позволяя модели динамически выделять связи элементов данных без жесткой локальности. С точки зрения здравого

смысла это ближе к человеческому взгляду — не к механическому сканированию, а к избирательному фокусу, где мозг инстинктивно связывает отдаленные детали в единую картину. Изначально они применялись для работы с базами данных, текстовыми данными и естественным языком. Как результат — появление новых архитектур больших языковых моделей (LLM). Но по сути те же слои можно задействовать для любых данных, в том числе для фрагментов изображений. В результате получается система, в большей степени приближенная к здравому смыслу и меньше зависящая от локальных переменных.



Такой подход эхом отзывается в заветах Дэвида Марра из его фундаментальной работы 1982 года, о которой я уже писал ранее, где машинное зрение трактуется на трех уровнях: вычислительном (цель — надежное извлечение информации), алгоритмическом (представления и процессы, включая иерархию от краев к объектам) и имплементационном (нейронные механизмы). Трансформеры вписываются в алгоритмический слой, усложняя вычислительную цепь через контекстуальные карты, предлагая абстрактное, масштабируемое решение, которое Марр, вероятно, приветствовал бы как шаг к постижению механизмов и обобщению подходов, а не к локальной эвристической подгонке параметров. В итоге машинное зрение приблизилось к тому, чтобы стать органичной частью универсального ИИ как основа единого когнитивного пространства, способного воспринимать мир целостно, — вопрос лишь в том, какой вычислительной ценой за это будет заплачено.

Один из соавторов работы «Attention Is All You Need» — Илья Полосухин — инженер и исследователь украинского происхождения, выросший в постсоветской инженерной среде. Он учился и начинал карьеру в Украине, после чего переехал в США и оказался в Google.

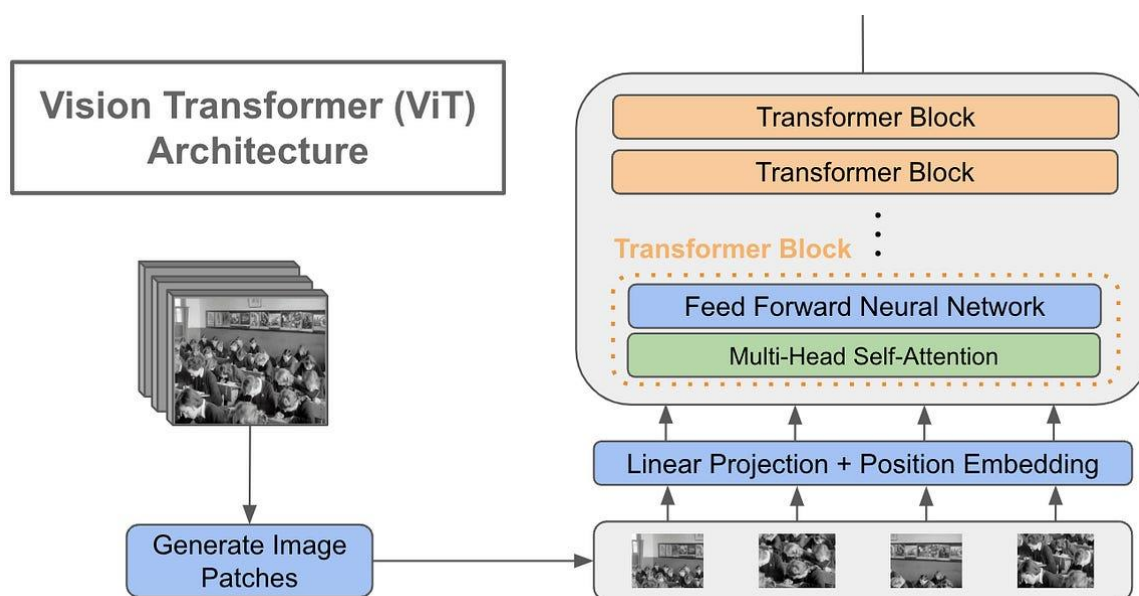
В 2026 году применение трансформеров в машинном зрении и восприятии видео достигло уровня, где архитектурные инновации фокусируются на эффективности и адаптации к реальным сценариям применения, постепенно переходя из разряда экспериментальных моделей к промышленным стандартам.

Vision Transformer — это продолжение развития задач классификации и детекции, пришедшее на смену ResNet или EfficientNet. На крупных наборах данных вроде ImageNet они показали 2–4% прирост по точности. Существует множество вариантов этой архитектуры для разных базовых задач. Так, для видеовосприятия ключевым прорывом стал Swin Transformer с его иерархической структурой и сдвиговыми окнами, минимизирующими квадратичную сложность до линейной, что позволяет обрабатывать последовательности кадров в реальном времени на периферийных устройствах — до 30 FPS на мобильных GPU, — с улучшением на 5–10% в задачах вроде распознавания движения на датасете Kinetics-400.

Другие передовые варианты, включая PVT (Pyramid Vision Transformer) с пирамидальными эмбедингами для мультишкального анализа и Twins с двойными ветвями внимания для пространственно-временных зависимостей, интегрируют 3D-видение и детекцию фейков, повышая mAP на COCO до 55% в комбинации с генеративными моделями. Значение порядка 55% mAP на COCO относится уже к верхнему уровню качества современных детекторов. По сравнению с моделями предыдущего поколения это означает прирост на десятки процентов относительно базового уровня и закрепляет переход к архитектурам, способным устойчиво работать с объектами разных масштабов и сложной сцены. Мультишкальный анализ означает, что модель одновременно рассматривает изображение на разных уровнях детализации: от мелких локальных фрагментов до крупных структур сцены. Это позволяет ей одинаково хорошо работать как с маленькими объектами, так и с крупными, объединяя точность и контекст. Эти архитектуры подчеркивают тренд унификации: трансформеры теперь сочетают понимание текста, изображений и видео в мультимодальных системах. Далее мы подробнее разберем самые характерные архитектуры.

ViT - Vision Transformer

В 2020 году компьютерное зрение получило мощный импульс от трансформеров — архитектур, завоевавших популярность в обработке естественного языка. Статья «Одно изображение стоит 16×16 слов» (An Image is Worth 16×16 Words) команды Google Brain под руководством Алексея Досовитского и соавторов, включая Лукаса Бейера и Александра Колесникова, представила смелый эксперимент: изображение разрезается на небольшие квадратные патчи размером 16×16 пикселей, каждый из которых преобразуется в вектор (аналог «слова» в тексте) с помощью линейной проекции, после чего вся последовательность подается в стандартный трансформер с механизмом самовнимания, полностью обходящимся без сверток. Это позволило модели захватывать глобальные зависимости между частями изображения на ранних этапах, в отличие от локальных фильтров нейростей сверточной архитектуры. При предобучении на огромном закрытом датасете JFT-300M (свыше 300 миллионов изображений) Vision Transformer не только доказал свою жизнеспособность, но и превзошел мощные базовые модели на ResNet: например, ViT-L/16 достигла 88,55% точности top-1 на ImageNet-1K, опередив ResNet-152 на 2–4% по ряду бенчмарков, включая перенос на задачи вроде семантической сегментации и детекции объектов. Обучение проходило стабильно, без типичных для глубоких сетей проблем с градиентами, а перенос обучения работал эффективно даже на меньших наборах данных.



ViT стал катализатором долгожданной тенденции к унификации представлений: механизмы внимания, изначально разработанные для последовательностей текста, теперь одинаково успешно применялись к визуальным патчам, сужая разнообразие «архитектурного зоопарка» и усиливая масштаб моделей. Граница между «NLP-мозгом» (обработкой

языка) и «CV-глазами» (анализом изображений) стиралась, открывая путь к мультимодальным системам, где один и тот же фреймворк справляется с текстом, видео и графикой. Неудивительно, что именно после ViT хлынул поток гибридов и преемников: от Swin Transformer с иерархическим и сдвиговым вниманием для эффективности до DeiT, оптимизированного для меньших датасетов через дистилляцию знаний. Однако у подхода имелся заметный минус: без предобучения на датасетах масштаба десятков или сотен миллионов изображений ViT уступал проверенным сверткам, демонстрируя худшую обобщаемость на малых объемах данных — подтверждение того, что эпоха больших данных оправдывает свое название, требуя не только вычислительных ресурсов, но и колоссальных объемов размеченной информации, для выигрыша в считанные проценты точности по сравнению с известными подходами.

В итоге ViT обозначил границы применимости: при наличии огромных данных и колоссальной аппаратной поддержки понимание мира алгоритмами трансформеров может работать независимо от модальности и для изображений, и для текста. И выигрыш по точности в несколько процентов скорее всего не предел, проблема только в непропорционально большей стоимости такого прироста в точности. За более пристальное внимание надо дорого платить — баланс между скоростью, качеством предсказаний, объемом данных и спецификой железа по-прежнему актуален, как и 50 лет назад.

Интересно, что Алексей Досовитский, один из пионеров унификации в машинном зрении, родился и вырос в Москве, где в 2009 году защитил кандидатскую и докторскую диссертации по математике в Московском государственном университете, специализируясь на функциональном анализе. Получив стипендию, он уехал в Германию, где в Университете Фрайбурга начал применять математический аппарат к задачам компьютерного зрения, разрабатывая алгоритмы для оптического потока и стереозрения. Уже в 2018 году, присоединившись к Google Brain, Досовитский стал соавтором Vision Transformer.

Другой Российский программист — Александр Колесников, тоже один из ключевых соавторов Vision Transformer, также начал свой путь в Москве, где в 2010-х годах получил степень магистра по математике в МГУ. Уехав в Европу, он защитил докторскую диссертацию по машинному обучению и компьютерному зрению в 2018 году в Институте науки и технологий Австрии (ISTA), сосредоточившись на системах, автоматически разбирающих визуальную информацию, что сразу выделило его как специалиста по глубоким представлениям и переносу обучения. Присоединившись к Google Brain и DeepMind, Колесников стал еще одним

соавтором прорывной работы по ViT в 2020 году. Его карьера продолжилась в OpenAI в качестве технического специалиста, а в 2025 году — в Meta.

В эпоху открытого исходного кода, зоопарк алгоритмов нейронных сетей — от сверточных конвейеров R-CNN до EfficientNet — превратился в общедоступный арсенал, где инновации появляются быстрее, чем успевают патентоваться. Например, у меня ушло больше года чтобы запатентовать свой алгоритм определения и визуализации траектории бильярдных шаров на бильярдном столе. Решающий импульс этому развитию придали открытые датасеты вроде PASCAL VOC, MS COCO и другие: они не просто предоставили тренировочные данные, а создали универсальные полигоны для соревнования, где метрики вроде mAP или AP стали объективными мерилами прогресса, помогая алгоритмам быстрее эволюционировать. VOC с его ранними классами объектов заложил фундамент сопоставимости в 2005, а COCO в 2014 расширил горизонты до многоплановых сцен с масками и подписями. Сейчас модели вроде ResNet или YOLO обучаются на миллионах аннотаций, а трансформеры на сотнях миллионов примеров.

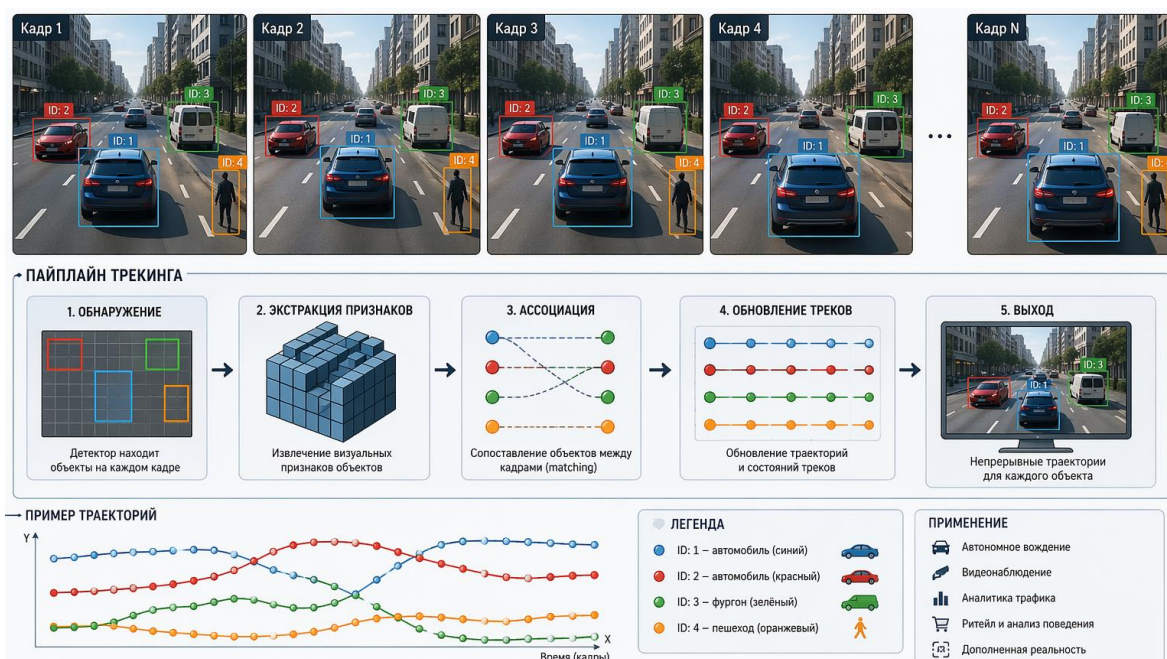
Истинные выгодоприобретатели в лице владельцев закрытых датасетов — гиганты вроде Google с JFT-300M или Meta с их закрытыми базами данных — обретают преимущество в нишевых задачах, параллельно продвигая открытые инициативы в качестве катализаторов общего прогресса. С другой стороны, платформы вроде Microsoft с их OpenCV и COCO продолжают доминировать в коллективном сознании сообщества: эти инструменты, бесплатные и масштабируемые, проникают в каждый проект — от прототипов на Raspberry Pi до промышленных систем видеонаблюдения, — монетизируют рост рынка не напрямую, а через экосистему, где небольшая доля от каждого внедрения (лицензии, облачные сервисы Azure) аккумулирует огромный капитал.

В итоге открытые датасеты и фреймворки продвинули машинное зрение, сделав его инфраструктурой, а не товаром, но именно эта доступность усилила разрыв: в мире, где данные — новая нефть, сообщество энтузиастов генерирует идеи и алгоритмы, а закрытые корпорации — эксплуатируют их, создавая продукты на базе больших данных и вычислительной мощности. Некоторую ценность пока что представляет процесс тщательного сбора, проверки, аннотации и организации данных: это не просто накопление изображений или видео, а их интеллектуальная обработка — фильтрация шумов, обеспечение разнообразия (по ракурсам, освещению, контекстам), устранение предвзятостей и создание метрик для оценки качества. Именно качественный датасет определяет, насколько модель обобщает знания, а не архитектура кода: без него даже сложная нейросеть остается слепой к реальности.

ЧАСТЬ 3.

Трекинг объектов

Переход от статического анализа изображений к трекингу объектов в видео — это более высокий уровень сложности в машинном зрении, поскольку меняется постановка задачи от единовременного распознавания к непрерывному мониторингу динамических элементов сцены. Трекинг объектов как класс задач подразумевает отслеживание идентичности и позиции одиночных (SOT) или множественных (MOT) предметов — будь то люди, автомобили или шары на столе — через последовательность кадров, с учетом их движения, взаимодействий и изменений внешнего вида, что требует не только детекции, но и временной согласованности этих детекций между кадрами. Общий пайплайн всех трекеров включает три ключевых этапа: инициализацию на первом кадре (определение начальной области интереса), предсказание позиции на очередном фрейме (на основе модели движения или корреляции) и обновление модели (ассоциация с предыдущими треками для сохранения идентификатора). Различия возникают в реализации этих шагов: одни трекеры полагаются на простые законы кинематики для предсказания (как фильтр Калмана или Венгерский алгоритм), другие — на обучение признакам (HOG или CNN), и визуальную ассоциацию, которая варьируется от сопоставления с известными метками до переидентификации эмбеддингов (ReID). Подробнее обо всем этом пойдет речь дальше.



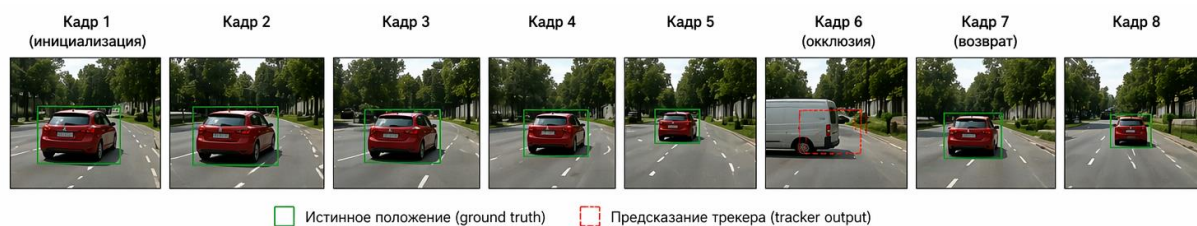
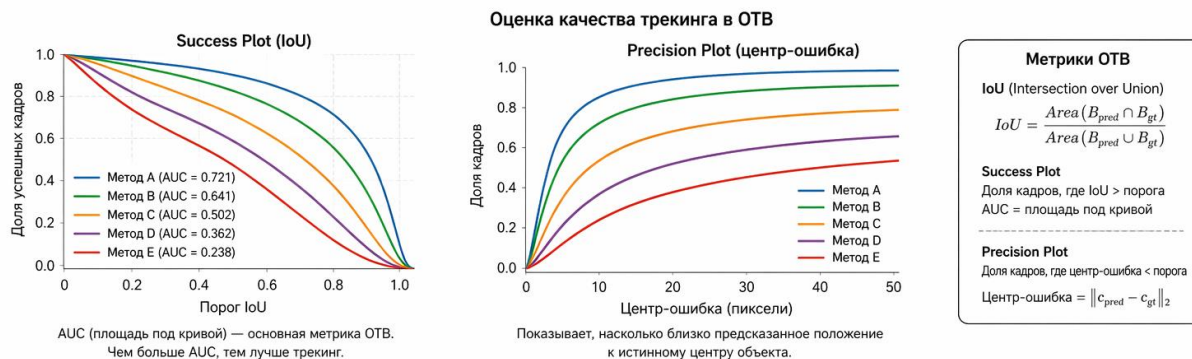
Самое сложное в трекерах — обработка окклюзий и переидентификаций, где потери информации приводят к каскаду ошибок. Осложняющим

фактором иногда является требование работать в режиме реального времени, что требует осознанного баланса между точностью и скоростью. Здесь же кроется пространство для изобретательства: варьируя признаки объекта (от цветowych гистограмм до глубоких сетей), модели движения (линейные или нелинейные) или стратегии ассоциации (простые пороги или глобальное внимание), можно адаптировать алгоритм под конкретные сценарии, от плавных траекторий в контролируемых видео до хаотичных толп в уличных трансляциях.

Как и у других задач по машинному зрению, у трекинга есть свои стандартные тестовые датасеты и методики оценки качества полученных результатов. Среди наиболее известных - Object Tracking Benchmark (OTB) и более масштабные LaSOT или TrackingNet. Далее мы подробнее на них остановимся.

Среди стандартных трекеров в библиотеке OpenCV выделяются MOSSE, MIL, TLD и MEDIANFLOW, каждый из которых воплощает уникальный баланс между скоростью, точностью и устойчивостью, адаптированный к различным сценариям видеоанализа. Подробнее мы поговорим про них дальше. MOSSE, основанный на корреляционных фильтрах, обеспечивает сверхвысокую скорость обработки для простых объектов с предсказуемыми движениями, минимизируя вычисления через спектральную корреляцию, но уступает в сложных условиях с деформациями или окклюзиями, где фокус на корреляции приводит к потере точности. MIL, опирающийся на множественные экземпляры обучения, демонстрирует превосходную устойчивость к частичным перекрытиям и шумам благодаря обобщению по "мешкам" положительных примеров, однако его итеративный поиск делает его заметно медленнее, что ограничивает применение в реальном времени. TLD сочетает трекинг, обучение и повторную детекцию для долгосрочного мониторинга, эффективно восстанавливая объект после потерь через адаптивный цикл положительных и отрицательных примеров, но подвержен дрейфу из-за накопления ложных обновлений, что снижает надежность в затяжных последовательностях. MEDIANFLOW, построенный на медианных оценках оптического потока, идеально подходит для плавных, линейных траекторий жестких объектов, игнорируя выбросы для стабильного предсказания, однако резко теряет эффективность при быстрых изменениях или нелинейных движениях, где медианные векторы не успевают адаптироваться. Такое разнообразие в OpenCV возникло для охвата спектра реальных задач: от высокоскоростного анализа в ограниченных условиях до устойчивого трекинга в шумных средах, позволяя разработчикам быстро выбирать оптимальный инструмент без внешних зависимостей и стимулируя эксперименты с гибридными комбинациями для достижения баланса между производительностью и точностью в конкретных приложениях.

Object Tracking Benchmark (OTB)



50+ видеопоследовательностей разной сложности



Разнообразные вызовы: окклюзия, изменение масштаба, движение камеры, освещение



Стандартный протокол оценки обеспечивает сравнимость методов



Результаты трекеров сравниваются по AUC и центр-ошибке

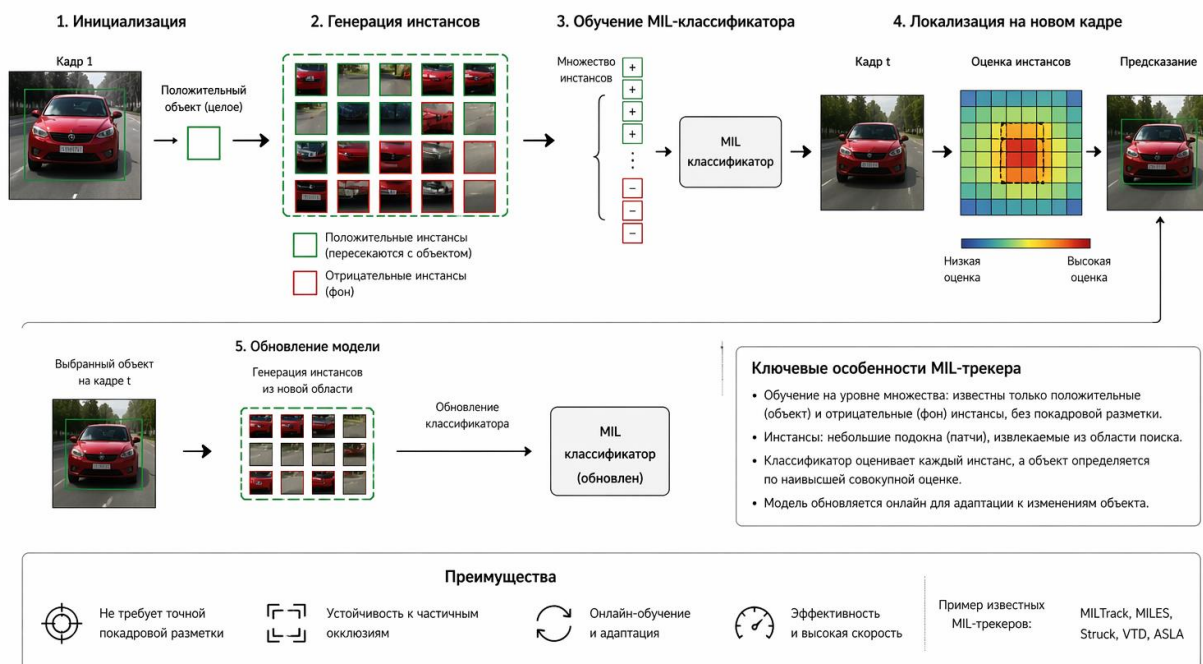
Object Tracking Benchmark (OTB) — это один из первых и наиболее влиятельных датасетов для оценки алгоритмов трекинга объектов в компьютерном зрении, разработанный в 2010 году командой исследователей из Калифорнийского университета Сан-Диего (США) под руководством Мин-Хсуана Янга (Ming-Hsuan Yang), Йи Ву (Yi Wu) и Джонгву Лима (Jongwoo Lim). Первая версия, OTB-50, включала 50 последовательностей видео с аннотациями в виде рамок для отслеживания одного объекта на кадр; в 2015 году вышла расширенная OTB-100, добавившая 50 новых последовательностей, всего 59 000 кадров с 11 атрибутами (например, окклюзии, изменения освещения, деформации, движение камеры и быстрая скорость объекта), что сделало его стандартом для сравнения трекеров.

Датасет фокусируется на трекинге одиночных объектов (SOT) в неконтролируемых условиях, с видео из различных источников, и использует две ключевых метрики: точность по центроидной ошибке, где идеал — 0 пикселей (precision plot) и точность по пересечению рамок (success plot). Точность по центроидной ошибке представляет собой кривую, иллюстрирующую зависимость процента успешных кадров по порогу ошибки: на оси X отображается центр ошибки (center location error, CLE) — расстояние в пикселях между предсказанной и истинной позицией центра рамки, а на оси Y — доля кадров, где ошибка не превышает

этого порога. Для расчета график строится по всем последовательностям видео, где трекер считается успешным, если CLE ниже заданного значения (обычно 20–50 пикселей), а средняя точность вычисляется как площадь под кривой или значение на фиксированном пороге. В отличие от точности по пересечению рамок, точность по центроидной ошибке акцентирует позиционную точность, что особенно полезно для задач с линейными траекториями, но может недооценивать случаи с деформациями объектов. Эта метрика позволяет объективно сравнивать алгоритмы, подчеркивая их эффективность в реальных сценариях с шумом и окклюзиями.

ОТВ не просто стал репозиторием тестовых видео, а превратился в катализатор эволюции трекинга, предложив сообществу строгие, воспроизводимые метрики оценки, что устранило хаос субъективных сравнений и стимулировало соревновательность среди разработчиков. Благодаря этому бенчмарку простые трекеры 1990-х уступили место более производительным системам, способным справляться с окклюзиями и деформациями. По состоянию на 2025 год ОТВ остается актуальным для базовых тестов, хотя не выдержал конкуренции с более масштабными, такими как LaSOT или TrackingNet.

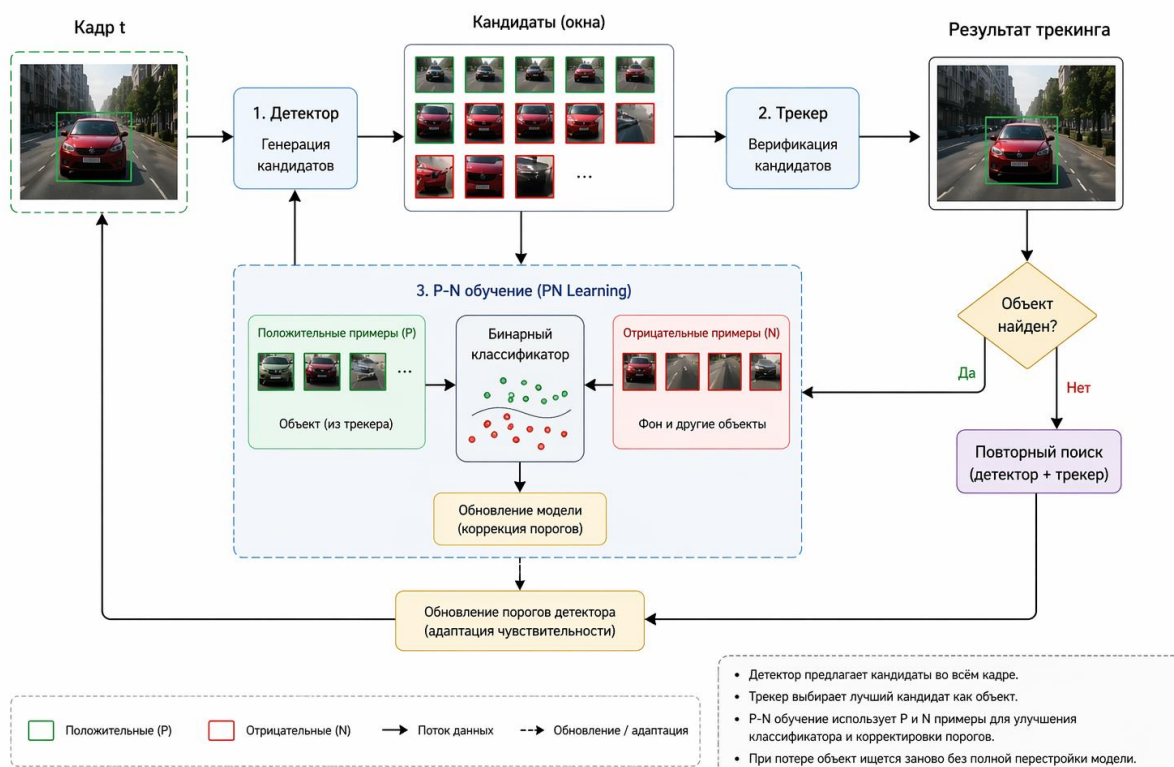
Трекер MIL



Трекер MIL (Multiple Instance Learning), интегрированный в библиотеку OpenCV с версии 3.0, представляет собой адаптацию парадигмы обучения на множестве объектных экземплярах. Он был предложен в 2009 году Борисом Бабенко из Университета Калифорнии в Сан-Диего в соавторстве с Мин-Хсуаном Янгом и Сержем Белонжи в работе "Robust Object Tracking with Online Multiple Instance Learning", опубликованной на CVPR. Этот подход стал ответом на вызовы трекинга в шумных видео, где традиционные жесткие методы на основе единственного примера объекта легко сбивались окклюзиями или изменениями освещения. Инженеры впервые применили вероятностный подход для классификации кандидатов в отслеживаемые объекты. Математически принцип прост и интуитивен: вместо обучения на одном «идеальном» патче объекта вокруг его рамки берутся множественные экземпляры — небольшие области внутри положительного «мешка», где метка положительна, если хотя бы один экземпляр содержит цель, а отрицательный мешок формируется из фоновых патчей; модель (обычно SVM или бустинг) учится обобщать, фокусируясь на наиболее дискриминативных признаках, как HOG или цветовые гистограммы, без жесткой локализации, что делает ее устойчивой к деформациям, но замедляет процесс за счет итеративного поиска по мешкам — до 10–20 FPS на процессоре против более 50 FPS у KCF трекера. На бенчмарках OTB этот трекер демонстрирует устойчивость к частичным перекрытиям объекта, достигая успеха в 60–70% сценариев с окклюзиями.

Борис Бабенко предложил один из первых практически устойчивых способов учитывать неопределённость разметки в трекинге. Его MIL-трекер не требовал точного знания, где именно находится объект внутри заданной области, и обучался на множестве возможных гипотез. Это сделало систему менее чувствительной к ошибкам инициализации и частичным сдвигам рамки. Подход не стал окончательным решением, но зафиксировал важную идею: в реальных задачах трекинга точная разметка редко доступна, и модель должна уметь работать с неопределённостью, а не игнорировать её.

Трекер TLD

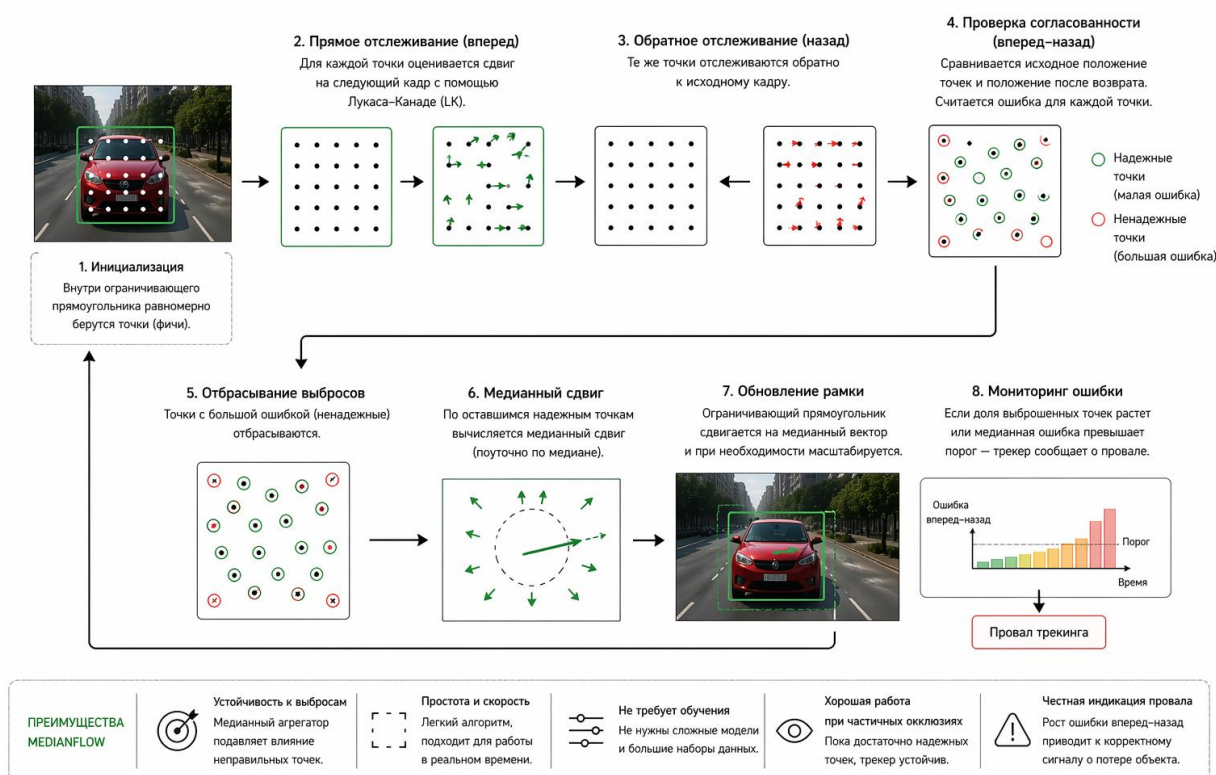


Tracking-Learning-Detection стал одним из первых адаптивных алгоритмов для долгосрочного отслеживания объектов в видео, предложенным в 2010 году на конференции ICCV чешско-британской командой под руководством Зденека Кала из Швейцарской высшей технической школы Цюриха, Кристиана Миколайчика из Университета Суррея и Йири Матаса из Чешского Технического Университета в статье «Tracking-Learning-Detection». Этот, можно сказать, революционный подход объединил три модуля: трекинг для предсказания позиции объекта на основе оптического потока (forward-backward error для оценки движения), детекцию для повторного поиска цели с помощью ферн-детектора на случайных деревьях решений и обучение для динамического обновления модели.

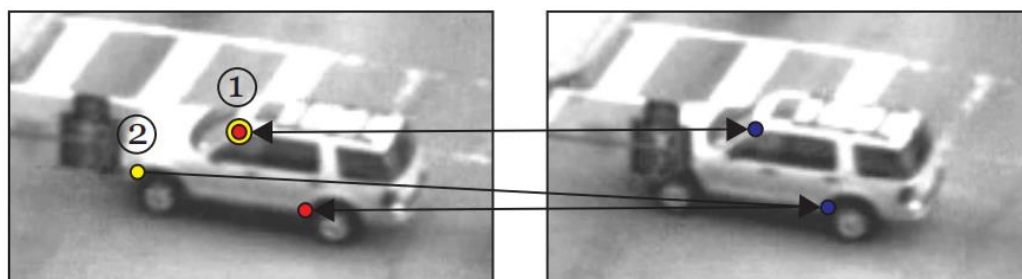
Алгоритм TLD опирается на цикл P-N обучения (PN Learning) — метод обучения, который использует положительные (P) и отрицательные (N) ограничения для обучения бинарного классификатора на основе помеченных и немеченных примеров. Цель — создать классификатор на основе заранее помеченных данных и улучшить его производительность на основе немеченных данных. Используются для баланса классификатора, где детектор генерирует кандидаты, трекер их верифицирует, а обучающий компонент итеративно корректирует пороги, чтобы объект «обнаруживался заново» при потере, минимизируя накопление ошибок без полной перестройки. В сравнении с MIL (Multiple Instance Learning),

TLD превосходит его в долгосрочном трекинге благодаря детекции, которая восстанавливает потерянные объекты быстрее, чем MIL, но склонен к дрейфу — постепенному смещению модели из-за ложных положительных обновлений, в то время как MIL лучше держит устойчивость к частичным окклюзиям за счет обобщения по «мешкам» данных, хотя и медленнее в итерациях.

Трекер MEDIANFLOW



Трекер MedianFlow предложили Зденек Калал, Кристиан Миколайчик и Йири Матас в 2010 году на конференции ICPR — International Conference on Pattern Recognition («Международная конференция по распознаванию образов») — в статье «Ошибка “вперед–назад”: автоматическое обнаружение срывов трекинга» («Forward–Backward Error: Automatic Detection of Tracking Failures»). Именно там авторы ввели простой критерий согласованности оптического потока вперед и назад и на его основе описали новый трекер, который они и назвали Median Flow.



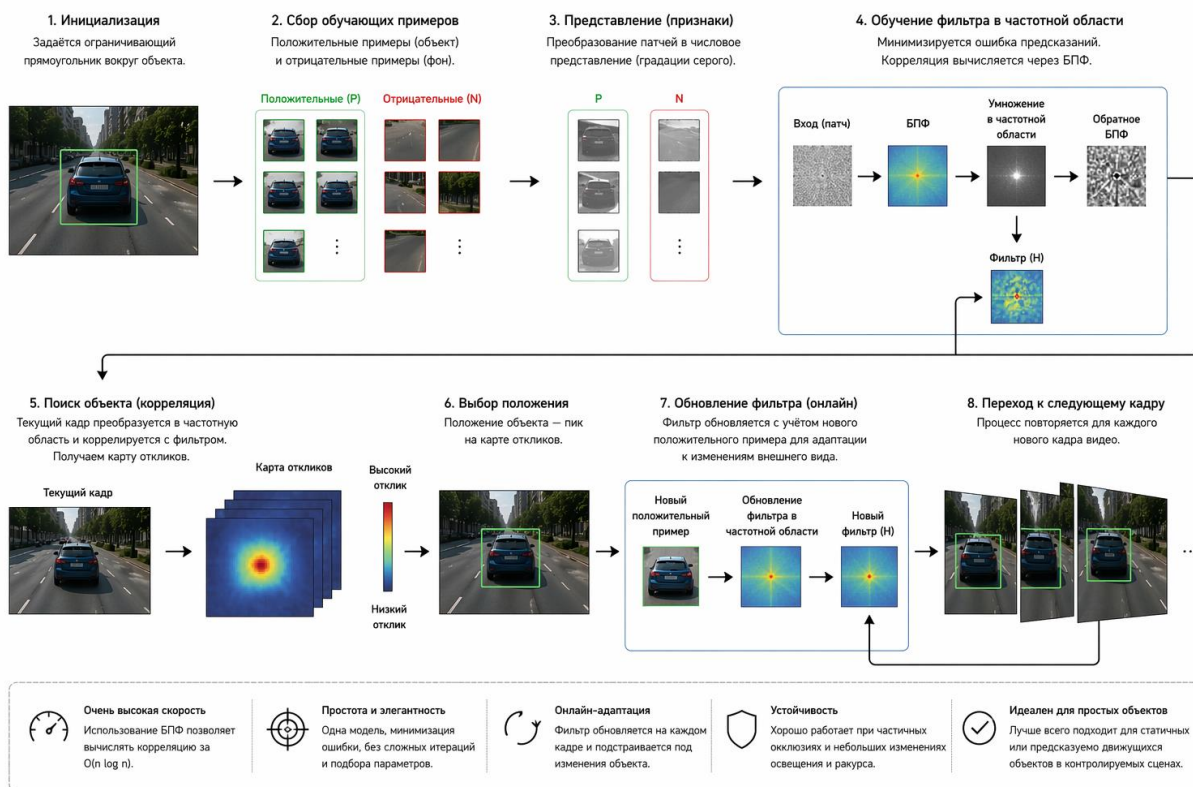
Идея в двух словах без формул такова. Внутри ограничивающего прямоугольника берут набор точек, для каждой оценивают сдвиг вперед по методу Лукаса–Канаде (Lucas–Kanade, LK), затем гоняют те же точки назад и сравнивают расхождение траекторий. Ненадежные точки отбрасывают, а движение объекта оценивают по медиане оставшихся сдвигов

— отсюда и название. Такой медианный агрегатор гасит «выбросы» и дает устойчивое приращение рамки до тех пор, пока объект виден и движется предсказуемо; при росте ошибки «вперед–назад» трекер честно сигнализирует о провале.

Главное достоинство MedianFlow — в его простоте. Он ничего не учит, работает поверх классического оптического потока Лукаса–Канаде и потому запускается мгновенно, в реальном времени даже на слабом процессоре. Медианная агрегация по точкам внутри рамки делает его устойчивым к выбросам: блик, локальная тень, случайная «плохая» текстура не тянут центр объекта за собой — большинство «здоровых» треков дают правильный вектор, и рамка смещается аккуратно. Встроенная проверка «вперед–назад» дает индикатор провала. MedianFlow предполагает небольшие межкадровые деформации и относительную стабильность внешнего вида: резкие повороты, сильный блюр при разбое, частичные/полные перекрытия объектом второго плана ведут к росту ошибки «вперед–назад» и к потере трека. Он плохо переживает долгие окклюзии и крупные изменения освещения; он не «переучивается» на новый вид объекта — в этом смысле он честнее, чем, скажем, MOSSE/KCF, но и менее «клеякий» при существенных изменениях шаблона. Поэтому MedianFlow часто используют как «короткий» трек между срабатываниями детектора или как компонент в системах класса TLD: трекер тянет ровные куски, индикатор срыва включает переобнаружение.

В сравнении с корреляционными трекерами (MOSSE, KCF) MedianFlow обычно проигрывает на сценах с крупными и быстрыми смещениями, но выигрывает по интерпретируемости и качеству самооценки: вы всегда знаете, когда доверять рамке, а когда — нет. На плотных, предсказуемых движениях (человек по пояс, предмет на конвейере, объект в кадре высокой частоты) он дает «ровную» траекторию и корректную локальную оценку масштаба при минимальной цене вычислений. Для задач с частыми перекрытиями и «рывками» лучше комбинировать его с периодическим детектированием и/или с трекерами, устойчивыми к окклюзиям, оставляя MedianFlow в роли быстрого, аккуратного звена «между» детекциями.

Трекер MOSSE



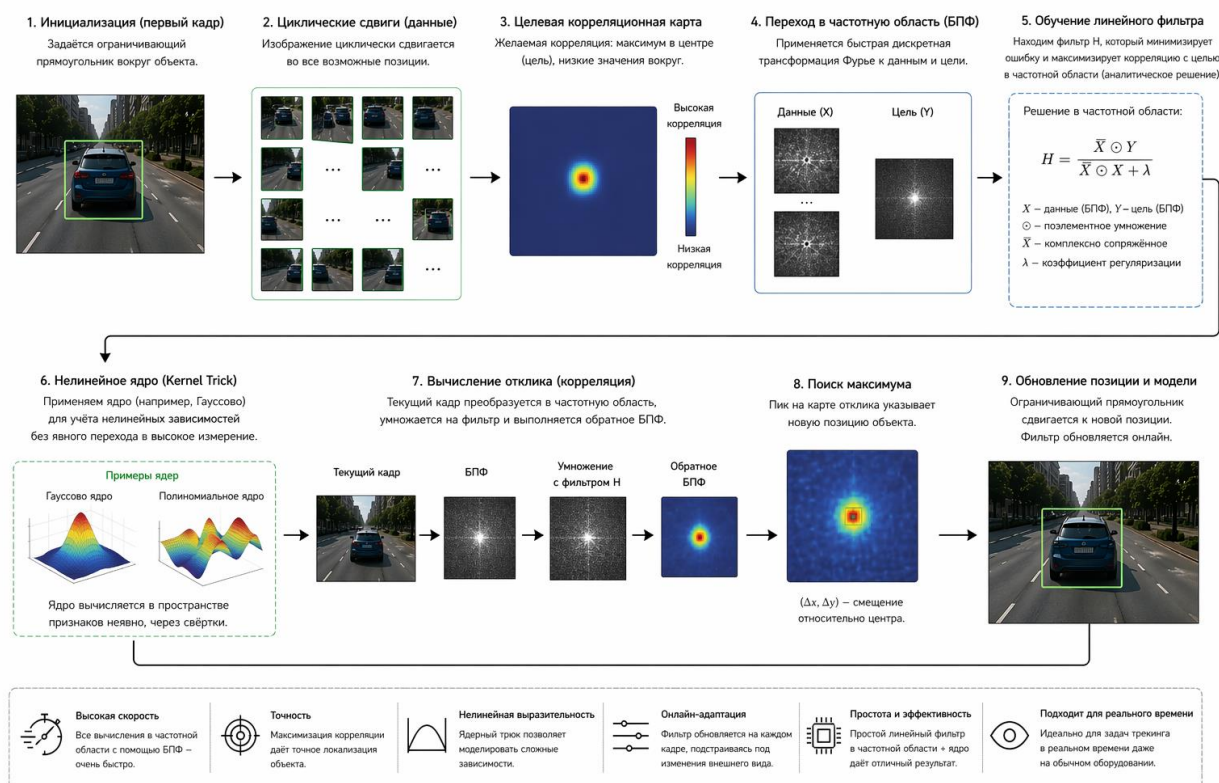
Трекер MOSSE (Minimum Output Sum of Squared Error) стал эталонным сверхбыстрым отслеживанием объектов в видео, предложенным в 2010 году на конференции CVPR командой из Университета штата Колорадо под руководством Дэвида Болме, Росса Бевериджа, Брюса Дрейпера и соавторами на конференции по компьютерному зрению и распознаванию образов — CVPR 2010 (Conference on Computer Vision and Pattern Recognition) в статье «Отслеживание объектов с использованием адаптивных корреляционных фильтров» (Visual Object Tracking using Adaptive Correlation Filters). Этот алгоритм достиг 669 кадров в секунду с точностью в 60–70% на простых сценариях. Он решил проблему задержек в раннем трекинге, обучая адаптивный корреляционный фильтр на первом кадре. Алгоритм MOSSE прост и элегантен: модель минимизирует общую ошибку предсказаний, обучая фильтр на положительных примерах объекта и отрицательных из фона, где корреляция в частотной области (через быструю преобразование Фурье) позволяет быстро вычислять отклики на сдвиги, обновляя фильтр онлайн для адаптации к изменениям, без сложных итераций — это делает его идеальным для простых объектов с предсказуемым движением, таких как статичные предметы в контролируемых видео. В сравнении с MIL (Multiple Instance Learning, 2009, Борис Бабенко и коллеги), MOSSE радикально быстрее (700 FPS против 10–20 у MIL) и проще для линейных траекторий, где корреляция эффективнее обобщает по "мешкам" экземпляров,

но уступает в устойчивости к окклюзиям, поскольку MIL лучше фильтрует шум через множественные положительные примеры; по отношению к MEDIANFLOW (2010, та же команда Болме и Дрейпера), MOSSE еще быстрее для жестких объектов (против 50–100 FPS у MEDIANFLOW) и менее чувствителен к выбросам в потоке, но полагается на корреляцию, а не на медианные векторы, что делает его уязвимее к нелинейным деформациям, где MEDIANFLOW держит форму лучше. В ретроспективе MOSSE подчеркивает ценность минимализма в трекинге, где скорость открывает двери для реального времени, становясь фундаментом для последующих корреляционных методов вроде KCF.

Алгоритмы первого поколения трекеров, такие как MOSSE, MIL, TLD и MEDIANFLOW, заложили фундамент практического отслеживания объектов в видео, опираясь на простые, но эффективные принципы — от корреляционных фильтров и медианных потоков до адаптивного обучения на множественных экземплярах, — что обеспечило высокую скорость и базовую устойчивость в контролируемых сценариях, хотя и с заметными ограничениями в обработке окклюзий, деформаций и нелинейных траекторий. К середине 2010-х годов этот подход начал трансформироваться под влиянием роста вычислительных мощностей и с появлением глубокого обучения: переход к следующему поколению ознаменовался интеграцией корреляционных механизмов с нелинейными ядрами, регрессионными сетями на основе CNN и дискриминативными оценками надежности, что повысило общую точность на 10–20% по бенчмаркам вроде OTB-100, сохранив при этом реальное время обработки. Эта эволюция отражает общий сдвиг в компьютерном зрении — от ручных признаков и локальных вычислений к обученным представлениям, способным лучше обобщать сложные динамики, — открывая путь к более устойчивым моделям, которые балансируют скорость и адаптивность в разнообразных реальных приложениях. Развитие базовых трекеров продолжается с применением более совершенных моделей, но их принципиальные последовательности остаются, поэтому полезно их понимать.

Трекер KCF

Трекер KCF (Kernelized Correlation Filters) представляет собой высокоэффективный алгоритм на основе корреляционных фильтров, предложенный в 2015 году португальскими исследователями Жоаном Ф. Энрикешем, Руи Казейру, Педру Мартиншем и Жорже Батистой из Университета Коимбры в статье «Высокоскоростное отслеживание с использованием ядерных корреляционных фильтров» (High-Speed Tracking with Kernelized Correlation Filters), опубликованной в журнале IEEE по анализу образов и машинному интеллекту (Transactions on Pattern Analysis and Machine Intelligence TPAMI). Этот подход, эволюционировавший из более ранних работ по корреляционным фильтрам (например, MOSSE 2010 года от Дэвида Болме и коллег), стал прорывом в скорости и точности для задач отслеживания объектов в видео, особенно в сценариях с линейными или предсказуемыми движениями, достигая до 100–200 кадров в секунду на стандартном процессоре.

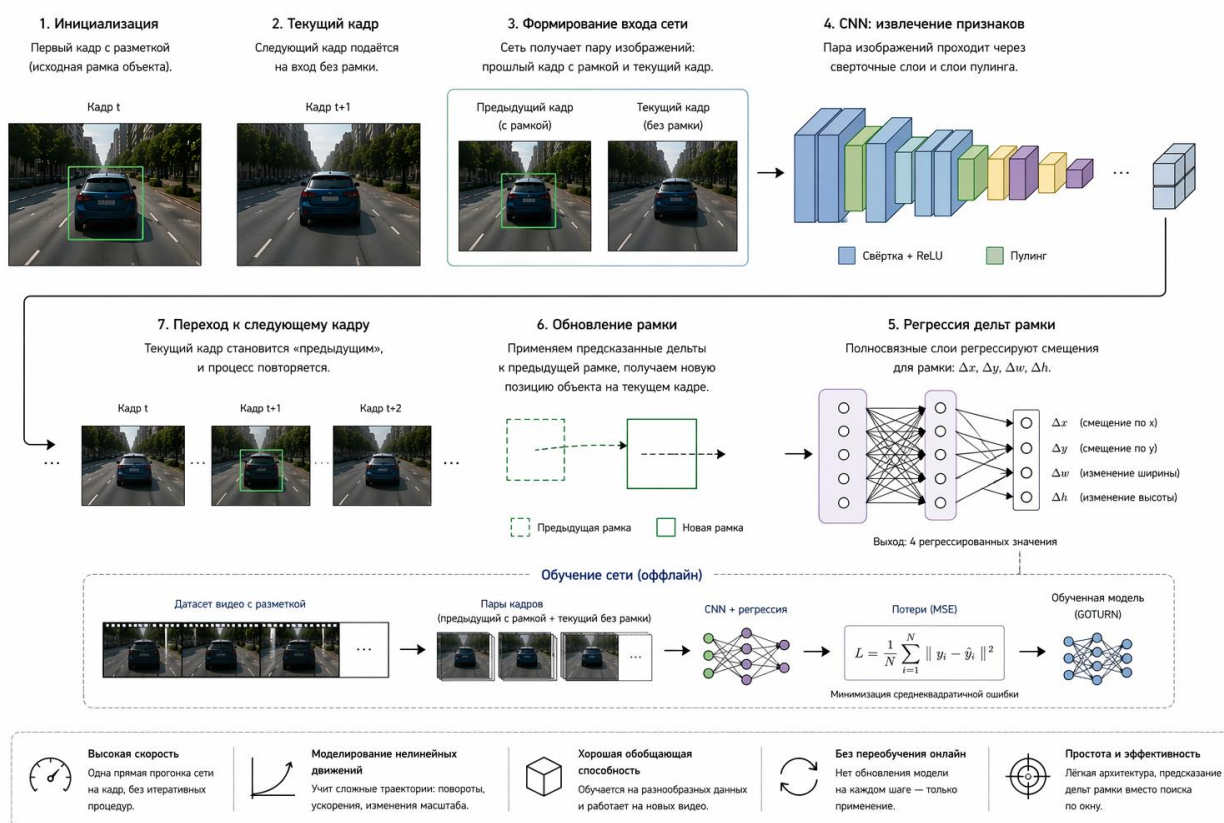


Математический принцип KCF опирается на идею обучения фильтра, который максимизирует корреляцию с целевым объектом в первом кадре: модель использует циклические сдвиги изображения для симуляции всех возможных позиций цели, обучая линейный фильтр в частотной области через быструю дискретную трансформацию Фурье, а затем обогащает его нелинейным ядром (например, Гауссовым или полиномиальным), чтобы захватывать сложные зависимости без явного вычисления в высокоразмерном пространстве — это позволяет быстро вычислять

отклики на последующих кадрах, находя пик максимальной корреляции как новую позицию объекта. В сравнении с трекером MIL, KCF выигрывает в скорости и точности для плавных траекторий (где MIL может тормозить из-за итеративного поиска по "мешкам" экземпляров), но уступает в устойчивости к частичным окклюзиям, поскольку MIL лучше обобщает через множественные положительные примеры, делая KCF идеальным для реального времени в контролируемых условиях, а MIL — для шумных сцен с неопределенностью. На бенчмарках OTB-100, KCF лидирует по FPS с успехом в 70–80% простых сценариев.

Трекер GOTURN

Трекер GOTURN (Generic Object Tracking Using Regression Networks) стал одним из первых сквозных решений на основе сверточных нейронных сетей для отслеживания объектов в видео и был предложен в 2016 году на конференции CVPR командой из Стэнфордского университета и Google под руководством Дэвида Хелда, Себастьяна Труна и Сильвио Савареze в статье "GOTURN: **Generic Object Tracking Using Regression Networks**". На бенчмарке OTB-100, трекер демонстрировал производительность в 65–75% на сценариях с нелинейными движениями. Этот подход отличался от предшественников тем, что предсказывал не только позицию, но и полное смещение рамки через регрессию, обучаясь на парах кадров без необходимости в дополнительных фильтрах.

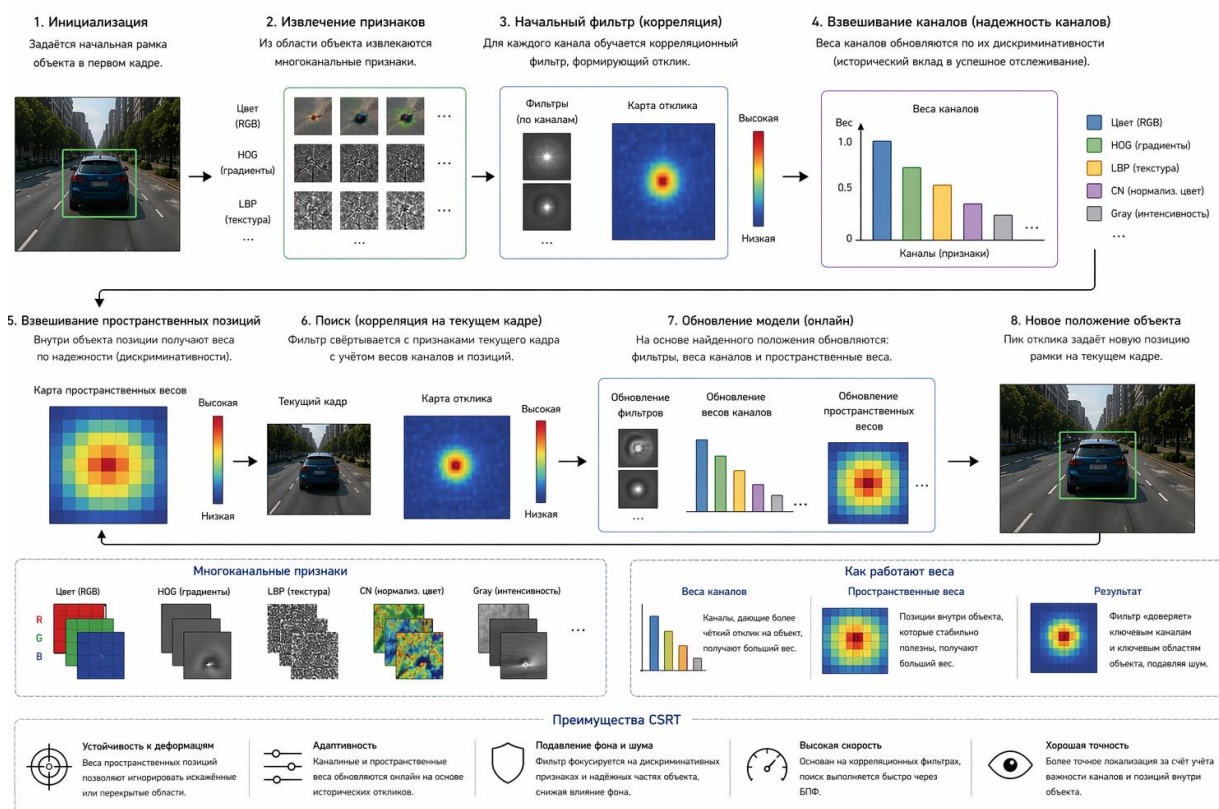


Математический принцип GOTURN опирается на регрессионную сеть: CNN принимает на вход два изображения — предыдущий кадр с исходной рамкой и текущий без него — и через последовательность сверточных слоев с пулингом извлекает признаки, которые затем регрессируют дельты координат (смещение по x, y, ширине и высоте), минимизируя среднеквадратичную ошибку для обобщения на новых кадрах; это позволяет модели адаптироваться к нелинейным траекториям, таким как повороты или ускорения, без переобучения на каждом шаге. В

сравнении с MIL (Multiple Instance Learning), GOTURN выигрывает в скорости и точности для динамичных сцен, поскольку регрессия находит не-локальные паттерны, в то время как MIL, фокусируясь на множественных экземплярах для устойчивости к окклюзиям, медленнее из-за итеративного поиска по "мешкам" данных. В отличие от KCF (Kernelized Correlation Filters), GOTURN лучше справляется с нелинейными движениями благодаря глубокому обучению, которое захватывает семантику, а не только корреляции, но уступает в простоте и скорости на процессоре (KCF достигает 200 FPS без предобучения), требуя внешней предобученной модели (на базе Caffe или Torch) и видеопроцессора для полной производительности. GOTURN — это хороший пример перехода к нейронным трекерам, но его устарелость (отсутствие обновлений после 2017 года) делают его нишевым инструментом для прототипов.

Трекер CSRT

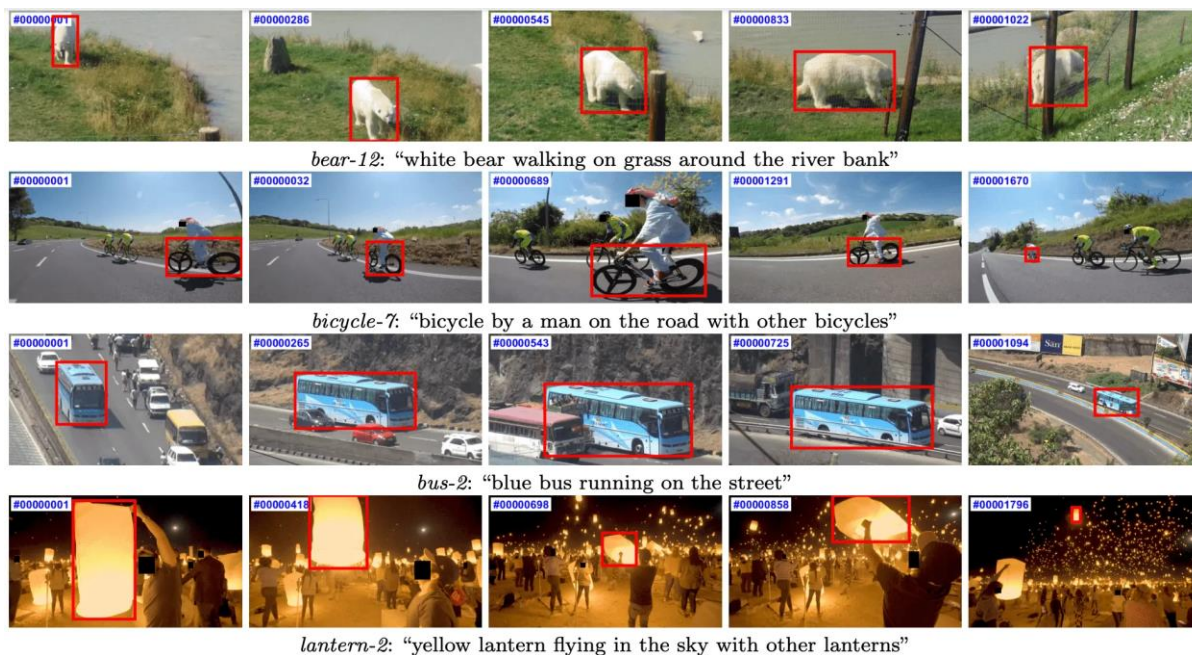
Трекер CSRT (Channel and Spatial Reliability Tracking) стал значимым продвижением в дискриминативных корреляционных фильтрах. Он был предложен в 2017 году на конференции CVPR (разработка начата в 2016) командой из Университета Любляны (Словения) и Чешского технического университета в Праге под руководством Алана Лукезича, Томаша Войиржа, Луки Чеховина Зайца, Иржи Матаса и Матея Кристана в статье "**Discriminative Correlation Filter with Channel and Spatial Reliability**". Этот алгоритм, подтвержденный бенчмарками OTB-100 и VOT-2016, где CSRT достиг производительности в 75–85% сценариев с деформациями объектов, решил проблему неравномерной информативности признаков в динамичных видео, балансируя высокую точность с приемлемой скоростью (10–20 FPS на CPU).



Математический принцип CSRT строится на расширении корреляционных фильтров: модель обучает фильтр не на равных признаках, а с учетом их надежности — веса для каналов (например, цветовых или градиентных, как в HOG) и пространственных позиций внутри объекта корректируются итеративно на основе исторических откликов, чтобы акцентировать дискриминативные области и подавлять шум от деформаций или фоновых помех, что позволяет фильтру адаптивно "доверять" ключевым частям цели без полной перестройки. В сравнении с GOTURN,

CSRT выигрывает в скорости и отсутствии зависимости от предобученной сверточной нейронной сети, не требуя видеопроцессора для базовой работы, но уступает в захвате сложных нелинейных траекторий, где регрессионная сеть GOTURN лучше обобщает семантику; по отношению к KCF, CSRT значительно устойчивее к деформациям благодаря механизму надежности, который повышает точность на 10–15% в изменяющихся сценах, хотя и медленнее за счет дополнительных вычислений по весам, и делает его оптимальным для задач, где баланс качества важнее чистой скорости.

LaSOT: Масштабный бенчмарк для долгосрочного трекинга объектов



Когда трекинг объектов в видео эволюционировал от простых фильтров к сложным нейронным сетям, датасет LaSOT (Large Single Object Tracking) стал настоящим поворотным моментом, подняв планку оценки алгоритмов до уровня реальных, затяжных сценариев. Предложенный в 2019 году на конференции CVPR, LaSOT был разработан как высококачественный бенчмарк для долгосрочного отслеживания одного объекта, где акцент сделан на последовательностях, имитирующих реальные практические задачи: объекты могут исчезать из вида, возвращаться через время, деформироваться или менять масштаб под влиянием движения камеры. Авторами работы стали Хэн Фан, Литинг Лин, Фан Ян, Пэн Чу, Гэ Дэн, Сицзя Ю, Хексин Бай, Юн Сюй, Чуньюань Ляо и Хайбин Линг, с существенным вкладом первых двух и Лингом в роли автора-корреспондента. Основной командой были исследователи из Университета Темпл (США), Южно-Китайского университета технологий (Китай) и лаборатории Meitu HiScene Lab (Китай), с поддержкой грантов от Национального ключевого плана R&D Китая, NSF США и Национального фонда естественных наук Китая. Эта многонациональная коллаборация — отличный пример глобального сотрудничества в компьютерном зрении, где китайские и американские учреждения объединили усилия для создания ресурса, доступного всему сообществу.

LaSOT возник как ответ на ограничения предшествующих бенчмарков, таких как OTB или VOT, которые фокусировались на коротких последовательностях (средняя длина 100–200 кадров) и часто стра-

дали от предвзятости категорий или низкого качества аннотаций. Нововведение LaSOT заключалось в создании крупнейшего на тот момент датасета для одиночного (SOT) трекинга: 1400 последовательностей видео общей длиной более 3,5 миллиона кадров (средняя длина — 2512 кадров, минимум 1000, максимум 11 397), собранных из YouTube под лицензией Creative Commons. Видео охватывают 70 классов объектов (от людей и животных до транспортных средств и дронов) с равномерным распределением — по 20 последовательностей на класс, — чтобы избежать категориальной предвзятости. Каждое видео аннотировано вручную: рамки на каждом кадре, с пометками отсутствия объекта при полной окклюзии или выходе из кадра. Уникальной фишкой стали 14 атрибутов вызовов (occlusion, deformation, scale variation, fast motion и т.д.), позволяющих детально анализировать слабости трекеров, а также естественные языковые описания для каждой последовательности, стимулирующие интеграцию визуального и лингвистического анализа. Аннотация проводилась командой из студентов и около 10 волонтеров, с двухэтапной верификацией: сначала группы аннотаторов, затем эксперты для качества, что обеспечило высокую плотность и точность меток.

Одним из главных открытий LaSOT стали новые метрики оценки, адаптированные для долгосрочного трекинга. Традиционные precision и success (IoU) дополнились нормализованной точностью, учитывающей размер объекта (площадь под кривой от 0 до 0,5 по порогам ошибки), чтобы избежать предвзятости к крупным целям. Протоколы оценки разделились на два: Протокол 1 (тест на всех 1400 последовательностях без обучения на LaSOT) для честного бенчмарка и Протокол 2 (80/20 сплит: 1120 для обучения, 280 для теста) для оценки эффекта переноса обучения. Оценка 35 трекеров (от классических вроде KCF до глубоких сиамских методов) выявила огромный спад производительности топ-моделей вроде MDNet или ECO, которая достигала всего 0,37 по precision и 0,41 по success, подчеркивая разрыв между краткосрочными тестами и реальностью. Это открытие — что трекеры "деградируют" на длинных последовательностях — стало катализатором и отправной точкой фокуса на долгосрочном трекинге.

Влияние LaSOT на сообщество было огромным: с момента публикации (arXiv:1809.07845, сентябрь 2018; CVPR 2019) статья набрала более 2000 цитирований (по Google Scholar на 2025 год), вдохновляя на десятки расширений, таких как LaSOT-Extension (2020) с 3000 последовательностями и LaSOT-Challenge (ежегодный конкурс с 2019). Открытый доступ позволил обучать сиамские трекеры вроде SiamRPN++, улучшив их на OTB на 2–5%, и стимулировал мультимодальные подходы с языковыми описаниями. Участники сообщества — от китайских лабораторий (SenseTime, Tsinghua) до американских (Temple University) — отметили LaSOT как «меняющий правила игры» для долгосрочного

трекинга. Атрибуты и метрики LaSOT стали стандартом, а его влияние ощущается в современных датасетах вроде GOT-10k или TrackingNet. В итоге LaSOT стал важным датасетом, продемонстрировавшим, что практический прогресс в трекинге нужно измерять не скоростью на коротких эпизодах, а реальной долгосрочной устойчивостью в потоке реального мира.

TrackingNet



В 2018 архитектуры уже научились ловить цель в динамике — корреляционные фильтры работали в сотнях кадров в секунду, сиамские сети уверенно «сопоставляли» патчи, — но данных для обучения по-настоящему глубоких моделей по-прежнему не было. Сообщество выкручивалось как умело: предобучало трекары на детекторных корпусах (ImageNet Video, YouTube-BB), где разметка редкая и время течет «рывками», а потом пыталось «доучивать» уже на коротких наборах вроде OTB или VOT. Тогда команда KAUST закрыла этот зазор одним мощным движением — представила TrackingNet, первый по-настоящему крупный датасет и бенчмарк «трекинга в реальном мире». Он включил свыше тридцати тысяч видеосегментов (около 140 часов), более 14 миллионов плотных аннотаций по кадрам и отдельный тест с примерно пятьюстами роликами (в публикации ECCV фигурирует точное число 511). Важная деталь — в каждом кадре размечается одна upright-рамка (ось-выравненная), что нормализует одино-объектный (SOT) трекинг без вращающихся боксов и сложных масок, а значит, делает сравнение трекаров воспроизводимым. Тест скрыт и оценивается через онлайн-сервер — это резко повышает качество соревнования и улучшает отчетность. Идея проста: если трекинг — это последовательность, учить его надо на последовательностях, много и разнообразно. Так у исследователей впервые появился серьезный датасет именно для SOT трекинга, а не детекции.

Откуда взялась такая масса видео? Авторы пошли путем «индустриального» датасета: стартовали с YouTube-Bounding Boxes, зачистили нежизнеспособные сюжеты, а класс «person», который занимал четверть аннотаций, аккуратно «расщепили» на подвиды по контексту,

чтобы распределение было богаче и реальнее. Главная инженерная находка — создание плотной разметки из редких кадров («один кадр в секунду»). Для этого они применили прием, знакомый каждому, кто когда-то «протягивал» разметку по видео: на коротких промежутках вперед и назад работает надежный дискриминативно корреляционный фильтр, например, KCF (Kernelized Correlation Filter), о котором написано выше. Затем его проходы усредняются с весами, чтобы получить устойчивую траекторию между редкими опорными метками. В результате тренировочная часть превратилась из «редкой детекции» в настоящий трекинг-корпус — с движением, частотами кадров, разными ракурсами и всеми радостями реального мира. На тесте — отдельные, вручную подобранные видео со сходным распределением классов и динамики, доступ к разметке закрыт: присылаете предсказания — получаете оценки. Такой «секвестр» закрывает вечную лазейку подглядывания в ответы и делает лидерборд осмысленным.

Что это изменило в практике? Во-первых, вместо предобучения на детекторах и донастройки на маленьких наборах появилось честное обучение на трекинге для трекинга: архитектуры стали смелее учитывать временной контекст, а не только видимость цели в одном кадре. Во-вторых, у сообщества появился единый эталон масштаба: более двадцати методов были сразу переоценены в одинаковых условиях, и стало видно, где у кого слабые места — быстрые перемещения, перекрытия, изменения освещенности. Наконец, и это важный «приземленный» эффект, — тонкая донастройка на небольшом куске TrackingNet давала прирост не только на самом тесте, но и на старых бенчмарках: авторы показывали прибавку порядка 1–2 процентных пунктов на OTB, что для зрелых задач заметно.

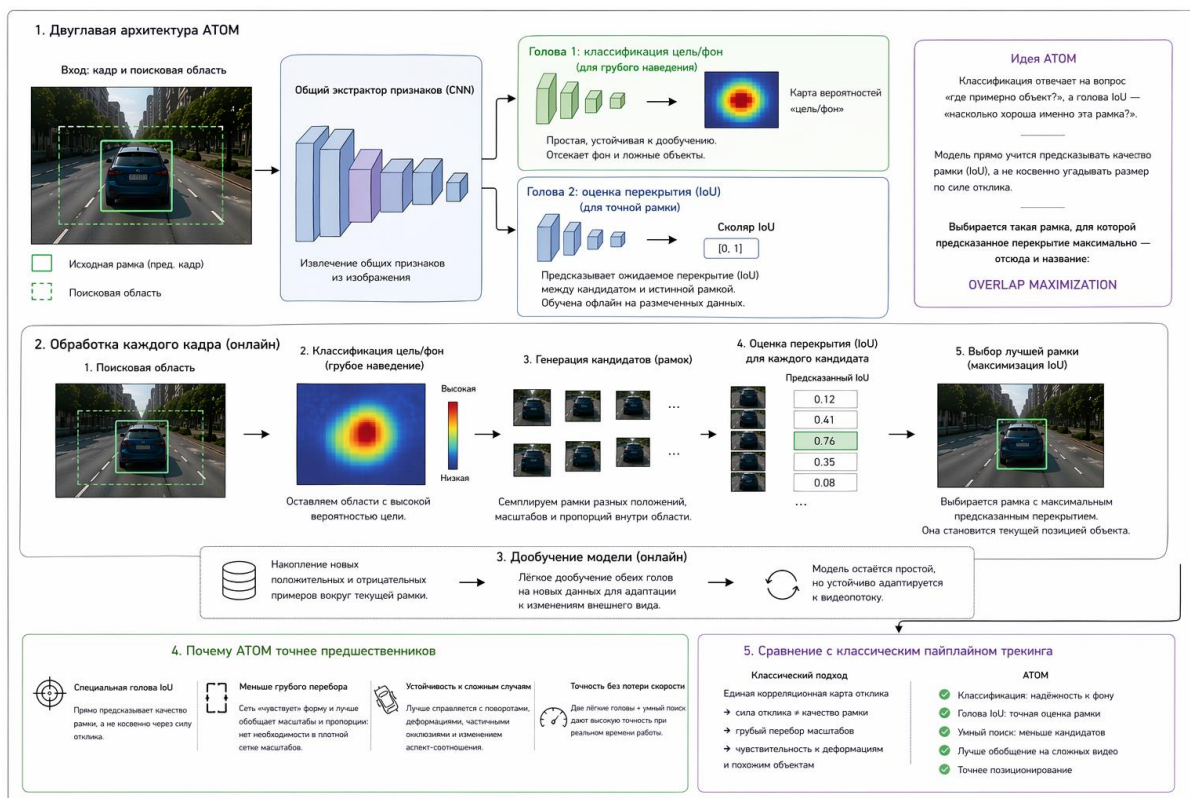
Маттиас Мюллер, Адель Биби, Сильвио Джанкола, Салман Аль-Субайхи и Бернард Ганем — все из университета науки и технологий имени короля Абдаллы (Саудовская Аравия) — сделали TrackingNet и выпустили его на ECCV-2018 вместе с онлайн-сервером и подробным сравнением с применяемыми тогда наборами (OTB, VOT, ALOV, UAV123, NfS и др.). Закрытый тест и единый сервер — стало фактическим стандартом для последующих бенчмарков.

Есть в этой истории и аккуратные оговорки. Во-первых, выровненные по осям рамки — компромисс ради воспроизводимости и скорости: они устраивают большинство приложений, но в задачах с сильными поворотами и деформациями маски или вращаемые боксы дадут большой выигрыш — поэтому рядом с TrackingNet выросли и «масочные» трекинг-наборы. Во-вторых, классическая однообъектная постановка — это осознанный выбор: она идеальна для доводки ядра трекера, но «мульт-

титрекинг» (MOT) требует своих датасетов и метрик. И, наконец, «трекинг в дикой природе» не отменяет доменной адаптации: если ваш продакшен — дрон-съемка или эндоскопия, без собственного дообучения не обойтись..

TrackingNet сделал для трекинга то, что ImageNet сделал когда-то для классификации: нормировал задачу. Именно поэтому почти любая новая работа по трекингу за последние годы, где бы она ни родилась, — так или иначе проходит через этот тест.

АТОМ



Трекер «АТОМ» означает точное отслеживание за счёт максимизации перекрытия — «Accurate Tracking by Overlap Maximization» — стал поворотом в одиночном трекинге: вместо того чтобы пытаться одним и тем же механизмом и «нащупывать» цель, и уточнять рамку, авторы разделили роли. Мартин Данельян, Гутам Бхат, Фахад Шахбаз Хан и Микаэль Фельсберг предложили двуглавую архитектуру: простая, но устойчиво дообучаемая **классификация «цель/фон»** для грубого наведения и отдельная **голова точной оценки рамки**, обученная офлайн предсказывать перекрытие (IoU) между кандидатом и истинной рамкой. Дальше: в каждом кадре модель ищет такую рамку, для которой предсказанное перекрытие максимально — отсюда и «overlap maximization». Именно внимательное, заранее выученное внимание к форме объекта дает АТОМ то, чего не хватало предшественникам: аккуратную рамку при поворотах, деформациях и изменении аспект-соотношения, без грубого перебора масштабов. Авторы подчеркивают, что это недостающий компонент классического пайплайна трекинга: классификация обеспечивает надежность к отвлекающим фон-объектам, а отдельная голова учит сеть именно оценивать качество рамки, а не косвенно угадывать ее размер по силе отклика карты признаков.

В одиночном трекинге (SOT) ResNet — как извлекатель признаков, и он почти всегда считается на каждом кадре в окне поиска вокруг прошлой позиции объекта. А вот шаблон (кусочек из первого кадра) обычно прогоняют через бэкбон один раз и кэшируют его признаки.

Как это устроено в популярных семействах:

— Сиамские трекеры (SiamFC / SiamRPN / SiamRPN++ / SiamFC++ и т. п.) в первом кадре считают признаки шаблона и запоминают их; в каждом следующем кадре они считают признаки поискового окна (обычно вырезка вокруг прошлой рамки), делают кросс-корреляцию «шаблон ↔ поиск» и получают карту отклика. Иногда шаблон слегка обновляют скользящим средним, но полного «детектора по всей картинке» там нет.

— АТОМ / DiMP: в каждом кадре бэкбон (часто ResNet-18/50) считает признаки для окна поиска; онлайн-классификатор «цель/фон» подстраивается по свежим примерам вокруг рамки, а отдельная «голова качества рамки» (IoU-оценка) помогает точно поставить бокс. Смысл тот же: каждый кадр — новые признаки для текущего окна, не глобальный прогон по всему изображению.

Исключения бывают в двух случаях. Во-первых, если трекер «потерялся», он может на пару шагов расширить область поиска вплоть до всего кадра — это не классический детектор, но поиск становится глобальным. Во-вторых, в мультиобъектном трекинге (MOT) почти всегда есть детектор на каждый кадр (YOLO/CenterNet/Detectron и др., часто тоже с ResNet внутри), а трекер лишь «сшивает» детекции между кадрами. Это другой подход: «детектируй каждый кадр и ассоциируй».

Итого: если речь про «обучаемые трекеры» в классическом SOT, ResNet/бэкбон работает на каждом кадре, но только на локальном окне; признаки шаблона из первого кадра кэшируются. Полный детектор «по всей сцене» кадр за кадром — это уже территория MOT или режимов «ре-детекта» при потере цели.

Контраст с аналогами прежних лет хорошо виден в сравнении. Корреляционные фильтры и их «гибриды» были сильны в устойчивом наведении, но почти всегда спотыкались о точную геометрию — обычно спасались грубым мультимасштабированием и теряли форму при изменении ракурса. Сиамская линия (SiamRPN/DaSiamRPN) добавила регрессию бокса и стала ощутимо точнее, но оставалась уязвимой при внеплоскостных поворотах и сильных деформациях: регрессионная голова не знала, **насколько** хороший бокс она вернула. АТОМ отыгрывает именно здесь: специализированная IoU-голова учит сеть различать «почти попали» и «попали точно», а значит — двигать рамку туда, где пере-

крытие действительно максимальное. В связке с онлайн-классификатором, который дообучается на текущем видео, это дает аккуратный баланс между устойчивостью и точностью.

Офлайн обучение — это все, что сеть учит заранее на большом датасете и потом приносит в продакшен в виде фиксированных весов. Онлайн — это то, что модель делает и/или доучивает уже «на месте», пока идет конкретное видео: кадр за кадром, без доступа к обучающей выборке и без долгих вычислений.

В трекинге это разделение особенно полезно. Например, в АТОМ офлайн учат отдельную «голову качества рамки» — сеть, которая по кандидату умеет предсказать, насколько хорошо он перекрывает истинный бокс (по сути приближает истину в последней инстанции). Это знание универсально и не зависит от вашего конкретного ролика, поэтому его разумно выучить раз и навсегда на больших данных и зафиксировать. А вот различать именно эту цель от именно этого фона приходится уже в конкретном видео: вокруг объекта — другие люди, другие фактуры, другой свет. Поэтому в онлайн АТОМ держит маленький, быстро подстраиваемый «классификатор цель/фон» и обновляет его по свежим примерам из текущего кадра. Дальше механика проста: генерируем набор кандидатов вокруг текущей позиции, прогоняем их через офлайн-выученную голову качества и выбираем тот, у которого предсказанное перекрытие максимально — это и есть «свести онлайн к поиску максимального перекрытия».

В DiMP идея та же, но они пошли дальше. Офлайн учат не только признаки и «чутье» к рамке, но и саму процедуру онлайн-подстройки: небольшой модуль-оптимизатор учится быстро находить хорошие параметры классификатора под конкретное видео за несколько итераций. А онлайн этот обученный оптимизатор уже применяет: на каждом шаге он берет свежие положительные/отрицательные примеры вокруг рамки и аккуратно обновляет «цель/фон», чтобы трекер не путался на фоне и не «дрейфовал» при смене вида объекта.

Для сравнения: сиамские трекеры почти целиком «офлайновые». Они заранее учатся мерить сходство между эталонным патчем и поисковым окном, а в онлайн никаких градиентов не считают: просто коррелируют «шаблон» с текущим кадром. Иногда обновляют сам шаблон (без обучения), но это все еще не онлайн-обучение в строгом смысле. Подробнее про них мы поговорим далее.

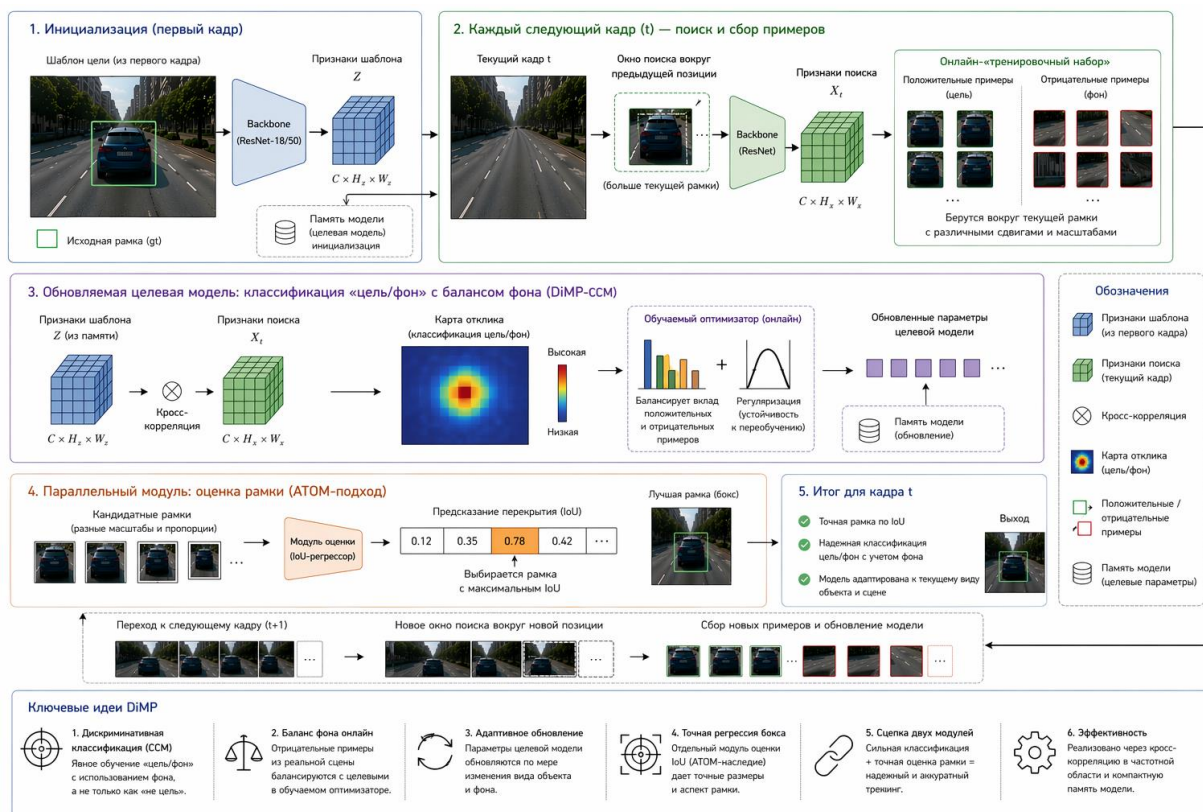
Зачем так делить? Потому что у офлайна и онлайн разные сильные стороны. Офлайн дает обобщение и воспроизводимость: модель видела тысячи сцен и знает, какие рамки бывают «точными» в среднем. Онлайн дает адаптацию: он подстраивается к вашей конкретной камере, свету, фону и к тому, как сегодня выглядит объект. Излишек онлайн —

риск «дрейфа» (модель перепутает цель с фоном), излишек офлайна — потеря чувствительности к вашей сцене. Поэтому современные трекеры держат «тяжелый» смысл в офлайне, а в онлайн-режиме решают только то, что действительно невозможно предусмотреть заранее: слегка подучить разделяющую границу и выбрать лучший из ближайших кандидатов.

По цифрам АТОМ в момент выхода задал новую планку на нескольких наборах сразу. На TrackingNet он показал «успех» (AUC) $\approx 70,3\%$ и прирост порядка 15% относительно предыдущих лучших решений — при скорости свыше 30 кадров/с на GTX 1080, то есть с практической латентностью для офлайн-аналитики и многих онлайн-сценариев. На LaSOT — одном из самых «длинных» и сложных бенчмарков — АТОМ достиг $\approx 51,5\%$ AUC, обогнав DaSiamRPN на 10 пунктов по «успеху» и существенно улучшив точность рамки. На VOT-2018 он лидировал по интегральной метрике EAO $\approx 0,401$, одновременно показывая лучшую точность среди топ-трекеров того года.

В инженерном смысле АТОМ стал «мостом» к следующему поколению. В АТОМ онлайн-классификатор еще обучается стандартной процедурой, а в DiMP ту же часть авторы сделали обучаемой целиком: не только что учим, но и как учим, — модуль предсказывает хорошие параметры классификатора под конкретное видео за несколько итераций, что заметно укрепляет адаптацию на длинных отрезках. В практической эксплуатации это различие чувствуется при постепенной смене внешнего вида цели, в «тяжелых» фонах и после кратковременных пропаданий. Но фундамент здесь — АТОМ, который впервые «вынес» точную рамку в самостоятельную голову и тем самым поменял стандарт для всего трекинга.

DiMP: «обучаемый оптимизатор» для трекинга (ICCV 2019)



DiMP — это трекер, представленный в 2019 году на конференции ICCV лабораторией компьютерного зрения Швейцарской высшей технической школы Цюриха: Гутамом Бхатом, Мартином Данельяном, Люком ван Гулом и Радуге Тимофте. Как я уже писал ранее, его ключевая идея - не просто онлайн-учить классификатор цели «как обычно», а научить саму процедуру обучения — то есть обучить модуль, который быстро предсказывает хорошие параметры целевой модели под конкретное видео. Такой «обучаемый оптимизатор» (в работе — итерационный steepest-descent-модуль, развернутый и обучаемый сквозным способом) и дает DiMP ту быструю и устойчивую адаптацию, которой не хватало предыдущему поколению трекеров. Код — в публичном репозитории [pytracking](https://github.com/yunqiangyao/pytracking).

Как это выглядит изнутри. На входе есть шаблон цели из первого кадра и поток последующих кадров. Сеть DiMP выдергивает признаки из первого кадра (обычно ResNet-18/50), строит для текущего видео целевую классификационную модель «цель/фон» и прогнозирует ее обновление по мере изменения вида объекта. Важный штрих — трекер явно использует фон: онлайн-«тренировочный набор» собирается из положительных и отрицательных примеров вокруг текущей рамки, а обучаемый

оптимизатор сразу балансирует их вклад. Параллельно работает модуль оценки рамки (унаследованный от АТОМ-подхода с предсказанием перекрытия), что дает точные размеры/аспект бокса. Итог — «сцепка» точного бокса и надежной классификации, которую модель умеет сама подстраивать под динамику сцены.

По цифрам DiMP в момент выхода уверенно задает планку сразу на нескольких бенчмарках. На TrackingNet версия DiMP-50 дает AUC 74,0% (против 73,3% у SiamRPN++ при том же ResNet-50), показывая прирост именно от лучшей онлайн-адаптации; на LaSOT — AUC 56,9% (выше АТОМ, особенно на длинных последовательностях); на GOT-10k — АО 61,1% и лидерство при обучении по «чистому» протоколу без внешних данных; на VOT-2018 — ЕАО 0,440, при этом отказов у DiMP заметно меньше, чем у скоростных сиамских моделей. Скорость при этом остается «практичной» — свыше 40 FPS на распространенном тогда железе.

Почему DiMP так быстро стал «базовым» в статьях и технологических стеках? Потому что он удачно попал в разрыв между двумя семьями методов. С одной стороны — сиамские трекеры: очень быстрые, но исторически плохо учившиеся от конкретного фона в конкретном видео (слишком «жесткий» шаблон сходства). С другой — дискриминативные онлайн-модели: гибкие, но чувствительные к тому, как именно вы их обучаете в онлайн. DiMP принес в эту вторую семью обучаемую процедуру обновления, сняв значительную часть ручной возни с шагами, регуляризациями и «костылями» под разные сцены. Отсюда и лавинообразное влияние: многие последующие работы — от PrDiMP (вероятностная регрессия рамки) до «супер»-вариантов с усиленным обучением — прямо строятся на DiMP/АТОМ-архитектуре, меняя головные части и подходы в обучении, но оставляя «обучаемую адаптацию» как стандарт де-факто.

Если подвести итог, DiMP — пример удачной оптимизации архитектуры. Он не прячет адаптацию в набор эмпирических правил, а делает ее частью сети и автоматически оптимизирует по будущим кадрам (в статье прямо показано, что такое обучение улучшает перенос на длинные ролики). На практике это обеспечивает меньше «срывов» на похожие объекты на фоне, аккуратное восстановление после кратких пропаданий и предсказуемое поведение на длительных архивах. И именно так DiMP записал свое имя в историю трекинга — грамотной архитектурой «обучаемой адаптации», которая стала основой для поколения трекеров 2019–2021 годов.

Сиамское семейство трекинга

Когда трекинг переходил из ранней эпохи «на лету подучиваем модель по видео» к эпохе больших данных и сверточных признаков, именно сиамские сети первыми показали, что можно вынести обучение из онлайн в офлайн и затем просто мерить сходство «эталона» с новым кадром. В 2016 Лука Бертинетто с коллегами представили SiamFC — «полностью сверточный» сиамский трекер: сеть заранее учится различать, где в большом поисковом фрагменте находится заданный образец цели, а во время работы все сводится к одной операции кросс-корреляции, которая дает тепловую карту: где «похоже» сильнее — туда и ставим рамку. Без подстройки весов по ходу видео и при скорости значительно выше реального времени это уже давало конкурентные результаты на используемых тогда бенчмарках — и зафиксировало сам алгоритм: шаблон (exemplar) + поиск (search) → карта отклика.

Дальше началась инженерия масштаба. Одной только «карты сходства» стало не хватать: нужно не просто найти максимум, а еще и точно угадать размер и пропорции рамки. В 2018 команда SenseTime и партнеров предложила SiamRPN — они добавили к сиамской основе голову из детектора Region Proposal Network. Теперь трекер в одном прогоне и классифицировал якорные прямоугольники «цель/фон», и оценивал для них сдвиги/масштабы; за счет этого резко выросла точность и устойчивость к изменению размера рамки, а скорость осталась «в реальном времени». На следующий год вышел SiamRPN++, который показал, как правильно «подружить» сиамский шаблон с глубокими бэкбонами вроде ResNet-50: авторы аккуратно разобрали проблему строгой трансляционной инвариантности, ввели «пространственно-осознанную» стратегию обучения и глубоко переработали корреляционные блоки (в том числе в сторону поканальной корреляции агрегации по слоям). Когда говорят о «пространственно-осознанной» стратегии, имеют в виду, что модель начинает явно учитывать геометрию и расположение признаков в пространстве. сама операция сравнения становится не просто «сверткой шаблона по поиску», а чем-то более чувствительным к структуре — например, поканальной корреляцией или её вариантами, где каждый канал несёт свою пространственную гипотезу. Модель начинает учитывать, что разные части объекта имеют разную роль. Центр, границы, углы — всё это обрабатывается по-разному. В некоторых архитектурах это выражается через отдельные головы для центра, размера, аспектного отношения; в других — через распределённые карты вероятностей. В тренировке начинают явно задаваться: где должен быть центр объекта, как выглядит корректное смещение, какие позиции являются фоном, даже если они похожи по текстуре. То есть модель учится не просто «узнавать объект», а понимать, где он находится и как он устроен в пространстве кадра. В итоге трекер выровнялся по качеству с лучшими

на OTB2015, VOT2018, UAV123 и LaSOT и при этом сохранил ключевую черту сиамской линии: скорость без онлайн-подстройки весов.

В конце 2018 — на CVPR-2019 — вышел SiamMask: к обучению добавили задачу побитовой сегментации и предсказание повернутых боксов. С теми же быстрыми сиамскими признаками модель научилась выдавать маску объекта в реальном времени, что резко расширило прикладные сценарии: робототехника, видео-редакторы и другие. Важно, что маска здесь — не добучение предыдущей архитектуры, а основная цель обучения, из-за чего сетка в принципе становится чувствительнее к точной геометрии.

Следующий поворот — отказ от «якорей». Якорные трекары (SiamRPN и наследники) опираются на набор прямоугольников с фиксированными аспектами и размерами; это удобно, но тянет за собой гиперпараметры которые надо подгонять под сцену. В 2019–2020 появились якорь-свободные сиамские трекары: SiamFC++ аккуратно разложил задачу на надежную классификацию и осмысленную оценку состояния цели (центр, размер) с практическими рекомендациями по тому, что и как оптимизировать.

SiamCAR и SiamBAN сформулировали классификацию и регрессию бокса в единой полностью сверточной голове, напрямую предсказывая положение и габариты объекта без перебора якорей. Практически это означало меньше настройки, лучше перенос между доменами и камерой, и, что немаловажно, стабильную работу на длинных видео без «расползания» рамки.

В целом алгоритмический «скелет» неизменен с 2016 года. Берется эталон цели из первого кадра. Сиамская сеть извлекает из него признаки и строит карту сходства «эталон ↔ поиск» как сверточную корреляцию. Поверх этой карты либо работает голова-детектор оценивает вероятность «цель/фон» и сдвиги для рамки, либо якорь-свободная голова напрямую предсказывает центр и размеры. Дальше идет слой преобразований: сгладить движение, подавить дубль-кандидаты (NMS), вернуть рамке правильные пропорции. В сильных версиях добавляют маску (как в SiamMask) или небольшое уточнение рамки по локальному градиенту.

Классические корреляционные фильтры (MOSSE-семейство, CF-ветка) были феноменально быстрыми, но страдали от грубой геометрии и зависели от ручного онлайн-обновления. GOTURN впервые показал, что можно «отрегрессировать» рамку одной прямой сетью на 100 FPS, но без карты сходства и детекторной головы его устойчивость в сложной динамике была ограничена. ATOM/DiMP добавили «умную» онлайн-адаптацию и точную оценку перекрытия, став сильнейшими на длинных треках. Сегодня границы размываются: многие промышленные

стеки держат два режима — сиамский «быстрый» и «тяжелый» с адаптацией — и переключают их по контексту.

Отдельно стоит сказать про влияние на сообщество и бенчмарки. С выходом TrackingNet и LaSOT стало видно, что сиамцы хорошо масштабируются: они учатся на последовательностях, а значит, получают разнообразие движений, перекрытий и ракурсов и при этом остаются простыми в работе. Именно поэтому в 2018–2020 годах почти любой новый трекер считал обязательным сравнение с SiamFC / SiamRPN / SiamRPN++ и их якорь-свободными наследниками по AUC/Precision на TrackingNet и по EAO на VOT. И это редкий случай, когда академическая «простота» совпала с практической: встраиваемым системам важны детерминизм и малые задержки, а сиамцы именно это и принесли.

Когда нужна высокая скорость, предсказуемая латентность и простота эксплуатации, сиамская семья — естественный первый выбор: SiamRPN++ как крепкий якорный базис, SiamFC++/SiamCAR/SiamBAN/Ocean, — если хотите меньше возни с гиперпараметрами и лучше перенос без «якорей», SiamMask, если важна форма. Когда ключевая задача — тонкая адаптация к фону и длинная память по ролику, имеет смысл смотреть в сторону ATOM/DiMP и их наследников — или гибридов, где сиамская скорость сочетается с легкой онлайн-подстройкой.

МОТ трекинг

В трекинге есть два ключевых направления — одиночный и многообъектный. Про SOT (single object tracking) мы уже говорили выше. Это персональное сопровождение: трекер не имеет задачи находить новые объекты, он должен удерживать заданный — через повороты, масштаб, перекрытия и выходы из кадра. Здесь выросли несколько семейств. Классика до глубоких сетей — корреляционные фильтры вроде MOSSE, KCF, которые брали скоростью, но проигрывали в геометрии. С 2016 года появились сиамские сети: SiamFC, затем более точные SiamRPN/SiamRPN++ с головой-предиктором объекта, масочные варианты вроде SiamMask и безъякорные SiamFC++/SiamBAN — все они учатся офлайн мерить «сходство шаблона и кадра» и потому держат высокую скорость без дообучения на лету. Меряют SOT по наборам OTB, VOT, а с 2018 — прежде всего по крупным TrackingNet, LaSOT и GOT-10k. Ключевые метрики — AUC «успеха» (по IoU), «точность» по центрам и их нормализованный вариант. Главные сложности — смена внешнего вида, жесткие перекрытия, дрожащая камера, похожие «дистракторы» рядом и необходимость быстро ре-детектировать цель после пропажи.

Мульти объектные трекаеры МОТ (multi-object tracking) выделяют в самостоятельный похожий класс задач. Принципиальная разница у них в том, что детектор объектов работает на каждом кадре (YOLO/Detectron/CenterNet и др.), а трекер «сшивает» детекции во времени, решая задачу соответствия. Базовые рабочие алгоритмы — SORT с калмановской динамикой и венгерским сопоставлением, DeepSORT с добавлением признаков «внешности» для ре-идентификации, затем более смелые схемы вроде Tracktor (использует регрессию бокса детектора), CenterTrack (учит смещения между кадрами), FairMOT и JDE (совмещают детекцию и эмбединги в одной сети), OC-SORT/BoT-SORT и ByteTrack, который показал, как аккуратный учет «низкобалльных» детекций резко снижает потери и ID-переключения.

Для МОТ есть свои бенчмарки: MOTChallenge (MOT16 / 17, MOT20), DanceTrack для сложной групповой динамики, KITTI / Waymo / BDD100K для дорожных сцен. Ключевые метрики — исторический MOTA, чувствительный к ошибкам детектора, IDF1 для качества идентичностей. Сложности МОТ — плотные толпы, длительные и полные перекрытия, срывы детектора из-за погоды и ракурсов, дрейф идентичностей и доменный сдвиг между городами и камерами; на практике качество почти линейно зависит от детектора, эмбедингов re-ID и дисциплины ассоциации при ограниченных задержках.

Практическое применение трекеров в бильярде

В бильярде я столкнулся с классической задачей MOT, только с особым поведением объектов: шары маленькие, одинаковые по размеру, часто перекрываются и время от времени размываются в вытянутые «сигары», если выдержка длинная. Поэтому я сразу пошел по пути детектора на каждый кадр и склейки результатов. Запускаю детектор вроде YOLO на каждом кадре, чтобы не терять шары в моменты столкновений и резких ускорений. Это не самый быстрый выбор, но это оправдано. Любые попытки «экономить» детекции через кадр тут же отражаются потерями и перескоками идентичностей особенно в быстрой фазе удара. Уже после детектора включается трекер: сбор детекций в траектории, сопоставление «кто есть кто» и аккуратная работа с пропажами.

Внутри трекера все держится на строгой и регламентированной механике. Для каждого шара создается модель почти равномерного движения — позиция и скорость в плоскости стола, которую прогнозирует фильтр Калмана. На новом кадре я открываю «ворота» вокруг предсказанной позиции и рассматриваю только те детекции, что физически правдоподобны: шар не может за одну тридцатую секунды телепортироваться на пол-стола. Этот физический барьер особенно важен при размытии: вытянутый след от быстро летящего шара детектор иногда видит, как два соседних «объекта», но предсказание Калмана помогает сразу отсеять ложный дубль и выбрать единственную правдоподобную точку. Когда кандидатов все же несколько, сшиваю прогнозы и детекции минимальными общими издержками — классическим назначением Венгера, где цена учитывает и расстояние, и размер рамки, и «внешность» шара.

Законы физики способствуют адаптивности модели. Фиксированное количество шаров, которые не появляются на столе из ниоткуда, и могут исчезнуть только в лузах. По умолчанию, моя система допускает разумные пределы ускорения. Вблизи бортов и при прогнозируемых столкновениях, я сознательно отпускаю фильтр: кратковременно увеличиваю допустимую вариативность скорости и разрешаю резкую смену направления. Сигналом служит либо приближение к линии борта (по координатам в плоскости стола), либо сближение двух треков: в этот момент я могу временно объединить их в «группу» и разрешить пересекаться без агрессивного штрафа, а уже на выходе из контакта аккуратно «развести» по траекториям с учетом предыдущего направления и положения. Такой режим спасает от типичной проблемы, когда два шара сливаются на пару кадров, и потом трекер меняет их ид имена местами.

YOLO на каждом кадре тяжеловат, но оправдан, поэтому я режу вычисления не за счет пропусков кадров, а за счет оптимизации кадра — вырезаю все лишнее вокруг стола.

Почему вообще пришлось писать собственный трекер? Честные MOT-схемы слишком доверяют re-ID и путают шары, сиамские SOT-трекеры не любят множественные соударения, а экономные пайплайны с редкими детекциями теряют ключевые кадры — период после удара и при столкновении. В итоге у получилась специальная адаптированная система, которая опирается на законы физики, соблюдает здравые ограничения движения и оптимально использует свои вычислительные ресурсы.

Как это выглядит на примере. Представьте: биток катится от короткого борта в центр, на пути — синий шар. На кадре 0 фильтр Калмана предсказывает белому ровную траекторию, синему — легкий сдвиг. На кадре 1 детектор отдает вытянутую «сигару» возле точки контакта и еще один круглый блик рядом — типичный смаз плюс блик. Анализаторы определяют вход в коллизионный режим, объединяют кандидатов в пару, а траектории продолжают по инерции. В склеивании траектории между кадрами запрещаем «перескок» идентичностей, если новое положение противоречит ожидаемому вектору модели шара. На кадре 2, белый отражается под острым углом — я это допускаю, потому что пересечение с бортом попало в «окно отскока», а скорость упала в разумных пределах. Синий чуть сместился и снова стал круглым на кадре — получаем уверенную детекцию, венгерский спокойно стыкует ее к «синему» треку. Ни один идентификатор не поменялся местами, оба трека пережили смаз и перекрытие.

RE-ID-трекинг

В многообъектном трекинге есть момент, когда геометрии мало. Прямоугольники сходятся, люди закрывают друг друга, объект выходит из кадра, а потом возвращается — и чистой кинематики уже не хватает, чтобы понять, «кто есть кто». Здесь и появляется ге-идентификация: способность узнавать один и тот же объект по его внешнему виду спустя время, через перекрытия и даже между разными камерами. В однообъектном трекинге это страховка от «дрейфа»; в многообъектном — возможная основа для аккуратной ассоциации; в мультикамерных системах — единственный способ «передавать» идентичность с одной камеры на другую.

Идея алгоритма проста: из каждой детекции извлекается вектор-отпечаток внешности — компактное числовое описание, устойчивое к поворотам, масштабу и свету. Практически это делает сверточная нейросеть с бэкбоном уровня ResNet-50: на обучении ее учат сразу двум вещам — различать личности (классификационная цель) и сортировать «одинаковые к одинаковым, разные к разным» (метрическое обучение с триплет- или контрастивной потерей). Отсюда и два привычных теста в мире re-ID: Rank-1 (как часто верный объект оказывается на первом месте при поиске по галерее) и mAP (насколько полно и точно мы возвращаем все экземпляры этой личности). Самые известные датасеты для людей — Market-1501, CUHK03, MSMT17; для транспорта — VeRi-776, CityFlow.

Как работает re-ID внутри MOT-пайплайна. На каждом кадре детектор (YOLO, CenterNet и тп.) дает кандидатов. Для каждого трека фильтр Калмана прогнозирует «куда он примерно приедет». Далее строится стоимостная матрица «треки ↔ детекции»: в нее входят два слагаемых — движение (насколько близко кандидат к прогнозу, с учетом ковариации) и внешность (близость эмбеддингов). По этой матрице венгерский алгоритм находит оптимальное сопоставление без конфликтов: каждому треку — не больше одной новой детекции, и наоборот. Если детекций не хватило, треки «живут по инерции» несколько кадров; если детекций много, лишние идут в «новые» или в очередь на проверку.

SORT — Simple Online and Realtime Tracking, «простое онлайн-отслеживание в реальном времени». В базовом виде он соединяет детектор, фильтр Калмана и венгерский алгоритм сопоставления. Классический DeepSORT ровно так и устроен: базовый SORT + re-ID-ветка, которая резко снижает перескоки идентичности в толпе. Современные варианты развивают ту же идею: JDE и FairMOT объединяют детектор и re-ID-ветку в одном бэкбоне (меньше конфликтов признаков, выше скорость); ByteTrack позволяет правильно учитывать «низкобалльные» детекции и не терять слабые, но настоящие объекты. Подробнее про них

мы ещё поговорим дальше. BoT-SORT использует противоположный подход: прикладные правила поверх эмбедингов — аккуратное временное и геометрическое сглаживание по внятным порогам нередко дают больше, чем очередной бэкбон.

Под прикладными правилами я имею в виду набор трезвых ограничений и процедур, которые наделяют трекер здравым смыслом. Он не «прыгает» к любой подходящей по внешности детекции, а сначала проверяет, возможна ли такая встреча геометрически и по времени. Предсказал Калман, что человек через 1/30 секунды сместится на полметра — значит, искать нужно в узком «воротнике» вокруг этой точки, а не на другом конце кадра. Камера дрожит — сперва компенсируем общий сдвиг фона, чтобы не приписывать его объектам. Кандидаты за пределами разумной скорости и направления отбрасываются до того, как включится ге-ID, иначе похожий эмбединг легко прилипнет к соседу. Сначала склеиваем только уверенные детекции с треками, потом аккуратно «добираем» слабые, не ломая уже сделанные связи. Порог для совпадения по внешности смягчаем во времени, чтобы одиночный блик или частичное перекрытие не переубедили трекер поменять идентичность. Новым трекам даем короткий период на подтверждение, пропавшим — возможность несколько кадров жить по инерции, если геометрия и контекст говорят, что объект вот-вот вернется из-за препятствия. Все это простые вещи по отдельности, но вместе они создают сложный порядок.

Re-ID незаменим там, где у объектов есть достаточно информативная внешность: одежда и аксессуары у людей, цвета и форма у машин, номера на борту судна. Он особенно силен при долгих перекрытиях и «возвращениях» — когда трек пропал и спустя секунды должен «подхватиться» снова. Но есть домены, где эмбедингу просто не за что зацепиться: однотипные предметы без текстуры (бильярдные шары, стандартные коробки, одноцветные каски) или сцены с жесткими бликами и регулярной подсветкой, где «внешность» скачет от кадра к кадру. Там лучше полагаться на физику, калибровку и строгую кинематику, а ге-ID использовать лишь как слабую подсказку — или вовсе отключать, чтобы не вносить ложную уверенность.

Можно назвать это инженерией «последней мили». У каждого трека есть память внешности — «галерея» последних эмбедингов. Ее нельзя слепо усреднять: ранние чистые портреты ценнее, чем поздние с частичными перекрытиями. Полезно хранить несколько «реперных» отпечатков и обновлять их только на уверенных сопоставлениях. Важны временные окна: после тяжелого перекрытия стоит на пару кадров «приглушить» роль внешности и довериться предсказанию движения, иначе два похожих человека просто обменяются идентичностями. Для камер с

разной оптикой и балансом белого помогает нормализация цвета и калибровка экспозиции; для мультикамерных систем — учет топологии: знание, какие зоны и в какое время может «достичь» человек с камеры А на камере Б, спасает не хуже сложной сети. И, конечно, собственное дообучение под задачу: даже отличная re-ID-модель, обученная на общих публичных наборах, ощутимо прибавляет, если вы ее слегка подстроите на своем домене: ракурсы, форма, одежда, свет.

Re-ID — очень важный элемент мультикамерных систем. В городских сетях десятки и сотни камер видят одного и того же человека под разными углами, в разное время и при разной погоде. Здесь re-ID становится центральной шиной: локальные MOT-треклеты стыкуются в общий «маршрут» по графу камер. На крупных соревнованиях вроде AI City Challenge без устойчивого эмбединга и аккуратной графовой ассоциации вы не соберете трассу сквозь квартал. Отдельно об этом соревновании тоже ещё поговорим ниже.

Подводя итог, внешность — это еще одна ось согласования, такая же важная, как движение; их сумма дает устойчивость, которой по отдельности не достичь. Но и наоборот: эмбединг — не магия. Он помогает, когда у объекта есть характерные признаки, и мешает при наличии множества похожих объектов. Поэтому, зрелые системы используют оба фактора, но настраивают их под прикладную конкретику своей задачи.

AI City Challenge

Если в истории компьютерного зрения ImageNet стал рубежом, где классификация изображений вырвалась из лабораторий в массовую технологию, то для трекинга похожую роль сыграл AI City Challenge — ежегодная серия задач и датасетов под эгидой воркшопа на CVPR. Первый выпуск прошел в 2017, а уже во втором, в 2018 году, организаторы — команда NVIDIA вместе с исследователями из университетов Мэриленда, Вашингтона, Олбани, Айовы и др. — предложили три прикладных направления: оценка скорости, детекция дорожных аномалий и переидентификация автомобилей. Смысл был предельно прагматичен: научить алгоритмы работать на реальных городских камерах, часто далеких от идеала по качеству, и мерить прогресс по метрикам, важным для дорожников и служб безопасности. Уже тогда на вызов откликнулись более семидесяти команд, а сами организаторы честно говорили, что вдохновлялись моделью ILSVRC, но для «умных городов» не хватало собственного, масштабного и чистого корпуса видео и единых правил оценки. Так появился формат, который ежегодно уточнялся и расширялся, но держал курс на прикладной эффект.

Решающий поворот случился в 2019, когда вместе с третьим выпуском представили CityFlow — первый действительно «городской» эталон для многообъектного и мультикамерного трекинга: синхронизированные HD-видео с сорока камер на десяти перекрестках, с калибровкой, геометрией и богатой разметкой. Задачи разделили равномерно: сквозной трекинг машин через сеть камер оценивали метрикой IDF1, которая фокусируется на сохранности идентичностей; переидентификацию по кропам из разных камер мерили по особому варианту mAP с усечением по топ-K, чтобы отражать реальные полицейские и диспетчерские сценарии «найди мне эту машину среди ближайших кандидатов». Важная деталь — требование к победителям публиковать код и любые добавочные разметки, чтобы результаты можно было воспроизвести и развивать, а не просто держать «секрет фирмы».

С тех пор CityFlow превратился в живую платформу: в 2021 году вышла обновленная версия CityFlowV2 с уточненной разметкой, расширенной географией и новыми тестовыми камерами; параллельно для трека по переидентификации разрешили и даже поощряли синтетические данные, например, VehicleX, чтобы командам было проще покрывать редкие модели и ракурсы. На этом же витке формата появился смелый трек: поиск траекторий машин по естественному языку — задача описать словами, что за автомобиль и куда он поехал, и система должна вернуть нужную траекторию. Все это подталкивало к междисциплинарности: трекинг плюс естественный язык, детекция плюс переидентификация, физика движения плюс визуальные признаки.

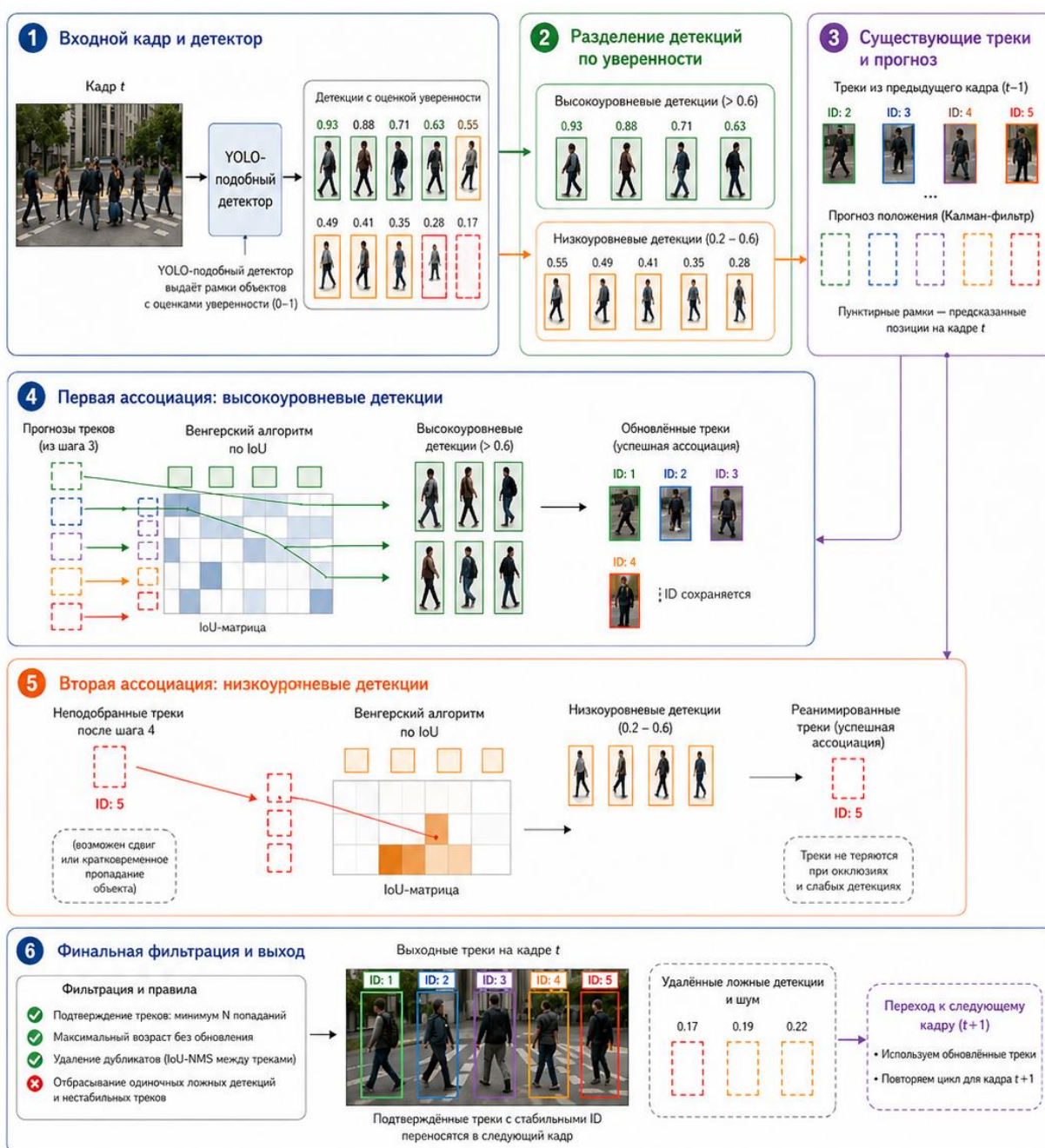
Пандемийные 2020-е не остановили рост, а даже ускорили его: воркшопы уходили в онлайн, но сами задачи становились богаче, а дорожные сцены — ближе к реальности: все чаще учитывались длинные коридоры из нескольких кварталов, сложные развязки, «дальняя геометрия» и мультикамерные связки с редкими перекрытиями во времени. Отчеты 4-го, 6-го и 7-го выпусков хорошо показывают эволюцию и организации, и участников: стандартом становится чистый протокол обучения/валидации, жесткие правила по повторяемости, а в текстах статей все чаще встречаются обсуждения не только точности, но и эксплуатации — от требований к вычислениям «на устройстве» до сценариев интеграции в диспетчерские.

В последующие годы горизонт еще больше расширился. К традиционным дорожным трекам добавились индустриальные сцены, складская логистика, безопасность работников и «цифровые двойники» на базе синтетики: в 2024 и 2025 в Challenge прямо заявлены наборы, сгенерированные в NVIDIA Omniverse, и новые формулировки задач — от 3D-мультикамерного трекинга людей и роботов до видео-вопросо-ответных задач по инцидентам на дороге. Это попытка закрыть реальный разрыв: не все данные можно собрать и разметить вживую, зато хорошая синтетика дает контролируемые условия и редкие сценарии — и, если метрики и протоколы выстроены правильно, она ускоряет путь до эксплуатации.

Если пытаться провести параллель с нашей общей темой книги, AI City — пример, как правильно формулировать цели. Не «сделайте трекер лучшим в вакууме», а «научитесь надежно отслеживать машину через десять камер, не путая ее с соседями; объясняйте, где и почему произошла остановка; ищите траекторию по словесному описанию; запускайтесь на реальном железе».

ByteTrack

ByteTrack: Multi-Object Tracking by Associating Every Detection Box, — это элегантный и высокоэффективный алгоритм для многообъектного трекинга (multi-object tracking, MOT), предложенный в 2021 году командой исследователей из Хуачжунского университета науки и технологий (Huazhong University of Science and Technology, Китай), Гонконгского университета (The University of Hong Kong) и компании ByteDance. Среди ключевых авторов работы — Ифу Чжан и Синган Ван и другие. Впервые представлен в 2021 году и опубликован на конференции European Conference on Computer Vision (ECCV) 2022.



В отличие от традиционных подходов, где низкоуверенные детекции (с вероятностью ниже порога, скажем 0,5) отбрасываются как шум, ByteTrack радикально меняет парадигму: он ассоциирует каждую детекцию с треками, разделяя процесс на два последовательных этапа, чтобы минимизировать потери объектов в сложных сценариях, таких как окклюзии, быстрые движения или плотные толпы. Архитектура строится вокруг высокоскоростного детектора — обычно YOLO, — который на каждом кадре генерирует рамки объектов со значениями уверенностей.

Затем применяется фильтр Калмана: он по прошлым положениям объекта предсказывает, где этот объект должен оказаться в следующем кадре. После этого начинается сопоставление новых обнаружений с уже существующими траекториями. Делается это как задача парного назначения: нужно решить, какая новая рамка относится к какому старому треку. Для оценки близости используется мера перекрытия рамок: чем сильнее предсказанная рамка трека пересекается с новой рамкой детектора, тем вероятнее, что это один и тот же объект. Само назначение выполняется венгерским алгоритмом, который находит наилучшее общее соответствие между всеми треками и всеми обнаружениями.

Сопоставление идёт в два этапа. Сначала система берёт только уверенные обнаружения — например, рамки с оценкой выше 0,6 — и связывает их с треками. В этот момент обновляются положения объектов и сохраняются их постоянные идентификаторы. Затем включается второй этап: к оставшимся «висячим» трекам, которые не нашли пару на первом шаге, пробуют присоединить слабые, но ещё не отброшенные обнаружения. Это могут быть рамки с низкой уверенностью, появившиеся из-за частичного перекрытия, размытия, плохого освещения или плотной толпы. Именно этот шаг часто спасает объект от потери: если человек или машина были временно видны хуже, трек не обрывается, а продолжает жить.

Ложные срабатывания отсекаются уже после сопоставления. Если низкоуверенная рамка не поддерживает ни один существующий трек, она считается шумом и удаляется. В этом и состоит сильная идея ByteTrack: использовать почти все обнаружения, но не доверять им одинаково. Уверенные рамки дают основной каркас, а слабые помогают не потерять объект в сложный момент. При этом алгоритму не нужен отдельный модуль переидентификации по внешнему виду и не нужны сложные графовые схемы. Достаточно предсказания движения, меры перекрытия рамок и аккуратного двухэтапного сопоставления.

Такое разделение даёт высокую скорость работы: алгоритм остаётся лёгким, хорошо подходит для обработки видеопотока в реальном времени и на обычной графической карте может достигать десятков кадров в секунду. Важнее всего то, что он решает одну из частых проблем

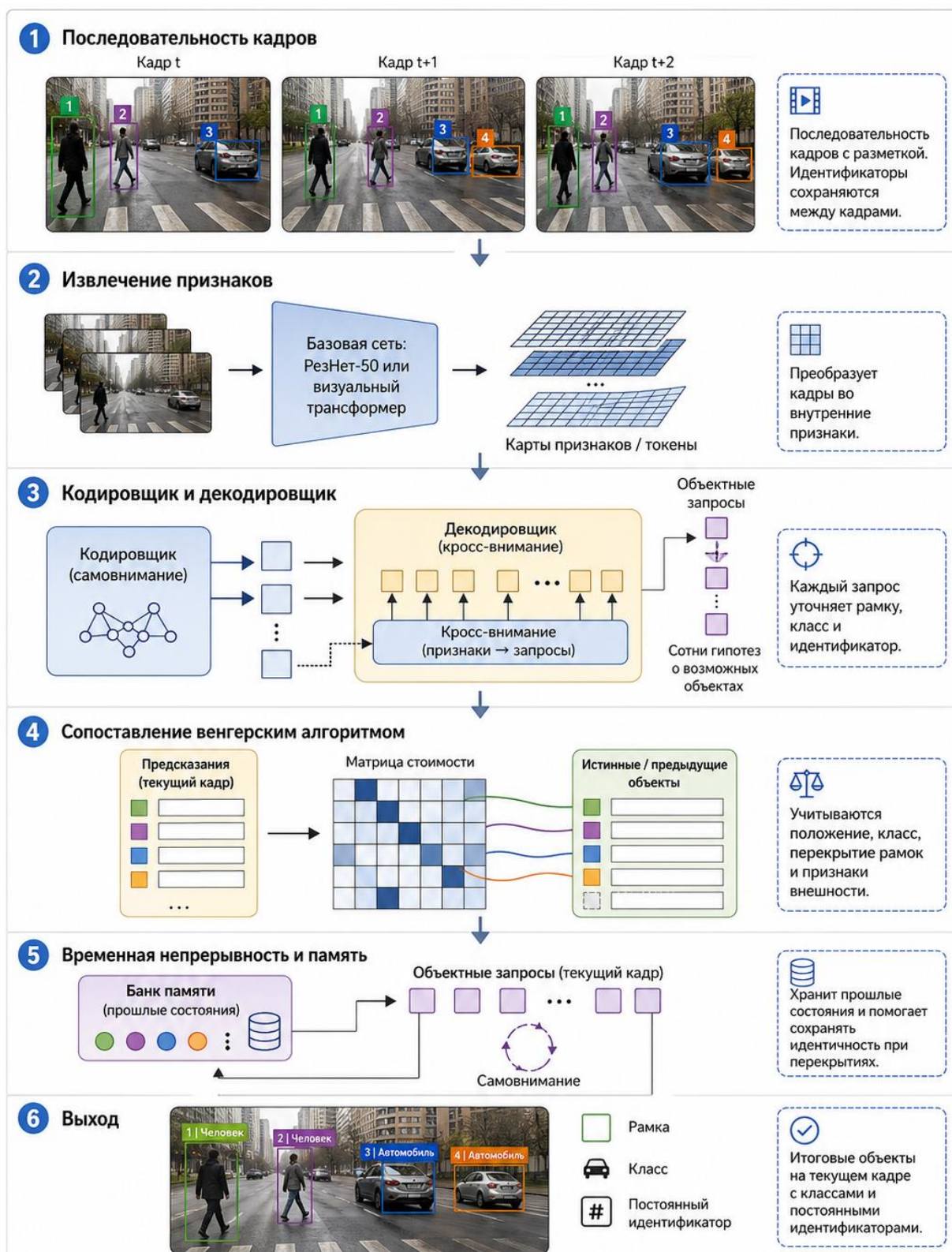
многокамерного и многолюдного трекинга: объект не исчезает из истории только потому, что детектор на одном кадре выдал по нему слабую уверенность. Это делает ByteTrack идеальным для реального времени. На бенчмарках вроде MOT17 и MOT20 модель устанавливает рекорды: HOTA (higher-order tracking accuracy) до 80,3% и MOTA (multiple object tracking accuracy) до 80,3% на MOT17, опережая предшественников вроде FairMOT на 5–10% в устойчивости к окклюзиям, при этом с минимальными вычислениями (всего 2–3 прохода на кадр).

MOTR

MOTR (Multi-Object Tracking with Transformers), предложенная в 2021 году (официально представлена на ECCV 2022) Фаньгао Цзэном, Бинь Дуном, Юань Чжаном, Тяньцай Ваном, Сяньюй Чжаном и Ичэнь Вэем (компания Megvii Technology и Шанхайский университет Цзяо Тун), представляет собой пионерскую сквозную архитектуру на базе трансформеров для многообъектного трекинга (multi-object tracking, MOT), где детекция и ассоциация треков объединяются в единую модель без внешних модулей, таких как фильтры Калмана или постобработка. В отличие от традиционных схем, где детектор вроде YOLO сначала отдельно находит объекты, а затем другой модуль связывает эти находки в траектории, трекер типа «трансформер для многообъектного трекинга» старается объединить обнаружение и сопровождение в одну общую модель. Он устроен по логике детекторного трансформера: сначала кодировщик извлекает признаки из кадров видео с помощью основной сети-извлекателя — например, ResNet-50 или визуального трансформера ViT. Затем эти признаки превращаются в последовательность внутренних представлений, которые поступают в декодировщик.

В декодировщике используется фиксированное число объектных запросов — обучаемых векторов, каждый из которых можно понимать, как гипотезу о возможном объекте или треке. Обычно таких запросов берут несколько сотен, чтобы модели хватало «мест» для всех потенциальных целей в кадре. Эти запросы сравниваются с признаками изображения через механизм перекрёстного внимания: каждый запрос как бы спрашивает у изображения, есть ли где-то объект, соответствующий его текущей гипотезе. На выходе модель сразу предсказывает рамки объектов, их классы и постоянные идентификаторы.

Ключевая идея состоит в том, что треки поддерживаются не внешним ассоциатором, а самими запросами. На первом кадре запросы ещё не имеют истории и сопоставляются с найденными объектами по наименьшей общей цене ошибки. Эта цена складывается из нескольких частей: насколько хорошо совпадают координаты рамок, насколько верно предсказан класс и насколько сильно предсказанная рамка перекрывает истинную. Для выбора наилучшего соответствия используется венгерский алгоритм, который распределяет предсказания по объектам так, чтобы суммарная ошибка была минимальной.



На следующих кадрах запросы уже несут память о предыдущем состоянии объекта. Они обновляются от кадра к кадру и снова сопоставляются с текущими наблюдениями. В цену сопоставления можно включать не только геометрию рамок, но и признаки внешности объекта: если два объекта похожи по внутреннему описанию, их косинусное сходство

будет выше, и модель с большей вероятностью сохранит тот же идентификатор. Это снижает перескоки номеров, когда люди или машины частично перекрывают друг друга, временно исчезают или проходят рядом.

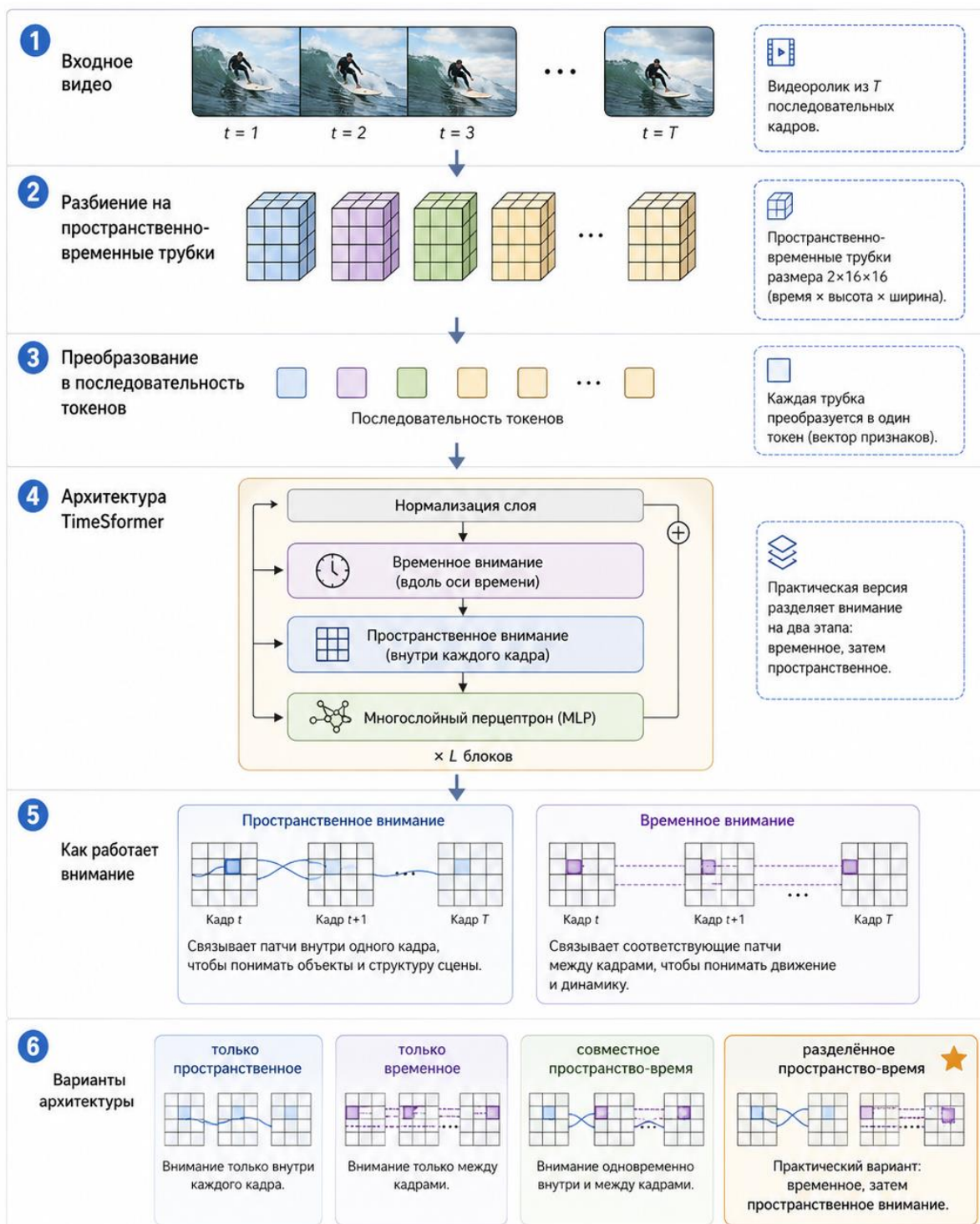
Для сохранения временной связности вводится банк памяти — буфер прошлых предсказаний и внутренних признаков. Он помогает модели удерживать историю объекта не только по одному предыдущему кадру, а на протяжении более длинного участка видео. Через механизм самовнимания текущие запросы могут обращаться к прошлым состояниям и уточнять, какой объект они продолжают сопровождать. Благодаря этому трекер лучше переживает короткие перекрытия и временные пропажи.

Обучается такая система сквозным образом: не по отдельности детектор, ассоциатор и модуль переидентификации, а вся цепочка сразу. Функция ошибки объединяет несколько задач: классификацию объектов, уточнение рамок и сохранение правильных треков во времени. Обычно обучение проводят на наборах данных для многообъектного трекинга, как например, на MOT17, а начальное умение обнаруживать объекты часто берут из предварительного обучения на крупном наборе данных COCO. В результате модель учится не просто видеть объекты в каждом кадре, а удерживать их как непрерывные сущности во времени.

Результаты впечатляют: MOTA до 72,7% и IDF1 до 74,1% на MOT17, опережая двухэтапные методы вроде FairMOT на 3–5% в надежности к пересечениям и скорости (до 15 FPS на V100 GPU). В контексте эволюции зрения MOTR подчеркивает силу трансформеров в унификации задач: от статической детекции к динамическому трекингу, где внимание глобально связывает кадры, делая модель масштабируемой для приложений вроде автономного вождения. В механизме внимания каждый элемент сравнивается со многими другими элементами, поэтому при росте числа объектов, запросов или признаков число сравнений быстро увеличивается. Для реального времени это становится проблемой: модель тратит много вычислений не только на сами объекты, но и на проверку всех возможных связей между ними. А оптимизируют это обычно так: уменьшают число объектных запросов, применяют внимание только в локальных окнах, оставляют в памяти не все прошлые кадры, а только ключевые состояния, отбрасывают слабые гипотезы до дорогого этапа сопоставления и считают внимание на укрупнённых картах признаков, а не на полном разрешении.

TimeSformer

TimeSformer, или Time-Space Transformer, — это инновационная архитектура на базе трансформеров, специально адаптированная для обработки видео, предложенная в 2021 году исследователями из Facebook AI Research (ныне Meta AI): Гедасом Бертасиусом, Хеном Ваном и Лоренцо Торресани.



Она решает ключевую проблему традиционных видео-трансформеров, таких как ранние модели на базе ViT, где последовательность патчей из всех кадров видео приводит к квадратичной вычислительной сложности, где N — общее число патчей (часто миллионы для длинных последовательностей), что делает их неэффективными для реального времени. Основная идея TimeSformer — разделение механизма внимания на пространственное и временное измерения, Чтобы сохранять общий контекст видеоролика и при этом не расходовать лишние вычислительные ресурсы, модель делит видео не на отдельные кадры целиком, а на небольшие пространственно-временные фрагменты. Каждый такой фрагмент охватывает небольшой участок изображения и короткий отрезок времени — например, два соседних кадра по участку размером шестнадцать на шестнадцать пикселей. Далее каждый такой фрагмент превращается во внутреннее представление, и из них собирается последовательность элементов, с которой модель работает примерно так же, как языковые трансформеры работают с последовательностью слов.

Дальше эта последовательность проходит через ряд одинаково устроенных блоков. В каждом блоке сначала выполняется нормализация слоя, затем работает механизм внимания, а после него — многослойный перцептрон, то есть небольшая полносвязная подмодель для дальнейшего преобразования признаков. Ключевая идея здесь в том, что внимание разделяют на две разные задачи. Первая отвечает за связи внутри одного кадра: она помогает понимать, какие части изображения относятся к одним и тем же объектам, как устроена сцена и где находятся важные области. Вторая отвечает за связи между кадрами: она сопоставляет соответствующие участки видео во времени и за счёт этого помогает улавливать движение, изменение позы и динамику действия.

Авторы рассмотрели несколько вариантов такой организации. В одном случае внимание работает только внутри каждого кадра и почти не учитывает временную структуру ролика. В другом — наоборот, модель сосредоточена в основном на изменениях во времени, а пространственная структура сцены упрощается. Есть и полный вариант, в котором пространственные и временные связи учитываются совместно, но он требует слишком много вычислений. Наиболее практичной оказалась раздельная схема: сначала модель обрабатывает временные связи для каждого пространственного элемента по отдельности, а затем — пространственные связи внутри кадров. За счёт этого вычислительная нагрузка растёт примерно пропорционально длине ролика, а не взрывается при увеличении числа кадров.

Именно поэтому такая архитектура остаётся пригодной на практике. Она может работать с относительно длинными видеопоследовательностями — порядка девяноста шести кадров — и на обычном графическом процессоре обрабатывать видео со скоростью в несколько десятков кадров в секунду. Это делает её полезной не только как исследовательскую конструкцию, но и как рабочий инструмент для анализа видео.

Обучение обычно строится в два этапа. Сначала модель предварительно обучают на крупных видеонаборах, например, на наборе Kinetics-400, где она учится в общем виде понимать движения, сцены и действия. Затем её дообучают под конкретную прикладную задачу — например, под распознавание действий человека в видео. Такой подход позволяет сначала набрать общий видеоконтекст, а потом уже приспособить модель к нужной предметной области.

TimeSformer достигает выдающихся результатов: 80,7% top-1 на Kinetics-400, опережая 3D-CNN вроде SlowFast на 2–5% при сопоставимых ресурсах. В контексте эволюции машинного зрения эта модель подчеркивает переход к унифицированным, масштабируемым подходам, где внимание это основа для понимания динамики окружающего мира.

ЧАСТЬ 4 АВТОНОМНЫЙ ТРАНСПОРТ

Одна из ключевых прикладных точек современного машинного зрения - автопилот для дорожного транспорта. Это самый сложный технологический вызов, двигающий прогресс целого множества технологических отраслей. Система должна быть одновременно чувствительной к деталям и устойчивой к шуму; «видеть далеко» и корректно работать вблизи; быстро реагировать и не допускать ложных срабатываний; масштабироваться по территориям при ограниченных ресурсах; использовать данные прошлого опыта, но не переобучаться на локальные шаблоны. Источники решений лежат в рациональном разбиении задачи на модули, в избыточности разнородных сенсоров и в явном учете неопределенности на каждом уровне — от детекции до принятия решений. Там, где неопределенность высока, система обязана снижать риск: понижать скорость, увеличивать дистанции, расширять «коридоры безопасности» или переходить к минимально опасной остановке. Это не частный выбор разработчиков, а требование к архитектуре, о становлении которой мы детально поговорим ниже.

Автопилот в инженерном смысле — это связанная система, где восприятие, прогноз и управление работают под контролем процедур безопасности. Входные данные дают камеры, лидары, радары, инерциальные датчики и спутники позиционирования; локализация и карты уточняют «где мы находимся» и «какова структура дороги»; алгоритмы восприятия выделяют разметку, объекты и свободное пространство; блоки предсказания оценивают вероятные траектории участников движения; планировщик выбирает допустимую траекторию с учетом динамики автомобиля, правил и ограничений сцепления; контуры управления доводят решение до руления, тяги и торможения. Все это упаковано в процессы диагностики, резервирования и деградации, а также в регламенты разработки и валидации (функциональная безопасность, безопасность задуманной функции, эксплуатационные процедуры и обновления ПО).

Идея автономного вождения возникла не из одного источника. Исследовательская линия идет от экспериментов с роботизированными платформами и первыми полевыми испытаниями в 1980–1990-х к системной постановке дорожной задачи в начале 2000-х. Идея практического внедрения обусловлена экономикой и безопасностью: снижение аварийности и тяжести последствий, дефицит водителей в логистике, рост городского трафика, требования к топливной эффективности и вы-

бросам, а также потребность производителей и операторов в предсказуемых сервисных издержках. Технологические драйверы — массовая доступность вычислений на GPU/ASIC, зрелость машинного обучения после 2012 года, появление крупных дорожных датасетов и симуляторов, удешевление и миниатюризация сенсоров, распространение высокодетальных карт и телеметрических платформ «флотного» обучения.

Технический стек эволюционировал от «алгоритмов по правилам» к обучаемым компонентам, но базовые роли сохранились. Восприятие перешло от отдельных детекторов в камерном изображении к мультимодальным моделям в «птичьем взгляде» (bird's-eye view, BEV) и сеткам занятости с калиброванной неопределенностью. Предсказание — от одиночной траектории к мультимодальным пучкам с вероятностями и оценкой ковариаций. Планирование — от вручную заданных шаблонов к гибридным решениям, где обучаемые модели сочетаются с жесткими ограничениями по безопасности и динамике. Параллельно идет упрощение «верхнего уровня» за счет более точных локализации и карт, а также усложнение «вертикали данных»: сбор, программная разметка, симуляция, регрессионные тесты, анализ отказов и непрерывное улучшение.

Стратегически индустрия делится на два проверенных подхода. В геофенс-службах — богатый сенсорный набор, локализация по HD-картам, строгие границы применения и контролируемые районы эксплуатации. В потребительских системах помощи водителю — преимущественно камерные стеки, опора на массовые данные и постепенное расширение функций при фиксированных скоростных и погодных ограничениях. Оба подхода признают ограничения операционного домена и строят «кейсы безопасности» как набор аргументов о допустимости работы в заданных условиях. Правовые рамки задают уровни автоматизации, процедуры сертификации и требования к ответственности при инцидентах.

Российская школа внесла свой вклад в развитие отрасли, особенно в решение проблемы сложных условий эксплуатации. Полевые проекты на зимних дорогах, роботакси в ограниченных районах, автономизация сельхоз- и промышленной техники, а также вклад в симуляцию, сенсорику и городское компьютерное зрение дали прикладные решения для множества вызовов: снег, пыль, неидеальная разметка, нестабильная оптика.

В Европе еще с конца 1990-х шли городские эксперименты с полностью автоматическими «киберкарами». Проект **CyberCars** (ЕС, 2001–2004) нацеливался на автономный общественный транспорт малого форм-фактора в городской среде; в связке участвовали INRIA (Франция), TNO (Нидерланды) и индустриальные партнеры, демонстрировались автономные проезды и разъезды на перекрестках с соблюдением

приоритетов. Это были реальные испытания автоматических шаттлов в ограниченных доменах (кампусы, бизнес-парки).

Наиболее прикладной пример — **ParkShuttle** под Роттердамом: автономная линия между метро Kralingse Zoom и бизнес-парком Rivium открылась еще в **1999** году и продолжала развиваться в 2000-х. Это была уже не демонстрация, а регулярная перевозка пассажиров на выделенной трассе с безводительским управлением. Академическая линия в Европе опиралась на опыт 1990-х (PROMETHEUS, VaMP/VITA-2) и итальянский проект **ARGO** (VisLab, Парма): в 1998 машина прошла ~2000 км по дорогам общего пользования с камерным зрением и одометрией.

В Японии действовала государственная программа **ASV (Advanced Safety Vehicle)**, фаза **ASV-3 (2001–2005)**. Минтранс (MLIT) системно финансировал и координировал разработки: межмашинные коммуникации, ассистенты перестроений и торможения, удержание в полосе, предупреждение о светофорах и знаках, связанное с инфраструктурой вождение. В 2004 на Всемирном конгрессе ITS в префектуре Айти крупные автопроизводители (Toyota, Nissan и др.) показывали такие системы публично.

В 1998 Mercedes-Benz вывел на рынок DISTRONIC (адаптивный круиз-контроль на радаре); в начале 2000-х разные производители внедряли LDW/LKA (предупреждение о сходе и ассистент удержания в полосе): первые коммерческие системы появились у Nissan/Infiniti и Toyota на японском рынке и в Северной Америке, затем у европейских марок.

В Великобритании к середине 2000-х оформляется университетский центр, позже известный как Oxford Mobile Robotics Group (Пол Ньюман): исследования локализации и карт «для города», последующие публичные испытания «RobotCar» и в итоге спин-офф Oxbotica в 2014. Это европейская школа «городского» восприятия и локализации.

Данные для обучения и оценки в этот период только формируются. В научном сообществе появляются тематические корпуса под отдельные задачи — например, Caltech Pedestrian (с 2009 г.) — и далее выходят дорожные «стандарты сравнения» вроде KITTI (2012), которые быстро становятся отраслевыми стандартами. Это важный сдвиг от закрытых датасетов к общим метрикам и публичным бенчмаркам.

Классификация автопилотов на основе уровней автоматизации была предложена Обществом автомобильных инженеров SAE (Society of Automotive Engineers) в 2014 году и описана в стандарте J3016. Она делит автономность вождения на 6 уровней — от L0 (отсутствие автома-

тизации) до L5 (полная автоматизация). Каждый уровень характеризуется степенью вмешательства водителя и степенью автономности системы.

L0 – Без автоматизации

На этом уровне система не предоставляет никакой помощи водителю. Водитель полностью контролирует машину, включая все аспекты вождения: руление, ускорение, торможение, а также принятие решений, связанных с безопасностью.

Пример: Современные автомобили с базовыми системами помощи водителю, такими как простые камеры для контроля за полосой движения, без какого-либо активного вмешательства.

L1 – Помощь водителю

На уровне L1 система может помогать в одном аспекте вождения, например, в ускорении или торможении, или рулении, но не одновременно. Водитель по-прежнему должен постоянно контролировать автомобиль и готов вмешаться в любой момент. Автопилоты на этом уровне включают адаптивный круиз-контроль или систему удержания в полосе, где система помогает регулировать скорость или следить за полосой, но водитель всегда контролирует все другие аспекты вождения.

L2 – Частичная автоматизация

На уровне L2 система может одновременно контролировать руление и торможение/ускорение. Однако, даже на этом уровне водитель должен оставаться внимательным и готовым взять на себя полный контроль в любой момент.

Пример: Tesla Autopilot, Mobileye, Honda Sensing. Система может одновременно рулить, ускорять и тормозить на дорогах, но водитель должен следить за дорогой и быть готовым вмешаться в случае необходимости.

L3 – Условная автоматизация

На уровне L3 система полностью управляет автомобилем в определенных условиях, таких как движение по автомагистрали, включая управление рулем, ускорением и торможением. Водитель должен оставаться готовым вмешаться по запросу системы (например, в случае сложных или неожиданных дорожных ситуаций). Водитель может отвести взгляд от дороги, но должен быть в состоянии быстро вернуться к управлению, если система подаст сигнал.

Пример: Audi Traffic Jam Pilot (представлен в 2017 году, но не был внедрен массово). Система может управлять автомобилем в пробках, но водителю необходимо быть готовым к вмешательству в сложных ситуациях.

L4 – Высокая автоматизация

На уровне L4 система может работать без вмешательства водителя в определенных условиях. Это может включать ограниченные географические зоны (например, на дорогах, где есть детализированные карты и четко понятные условия движения). Водитель не обязателен для контроля, и система способна взять на себя полный контроль, включая все аспекты вождения, даже если водителя нет в автомобиле.

Пример: Waymo (проект Google) в 2018 году представил роботы-кары в рамках L4 — автомобили без руля и педалей, которые могут полностью функционировать в геофенсированных областях (например, в районе Финикса, штат Аризона), без вмешательства водителя.

L5 – Полная автоматизация

На уровне L5 система способна работать без водителя в любых условиях. Водитель не нужен вообще, автомобиль может самостоятельно управлять всеми аспектами вождения, включая восприятие, принятие решений, планирование маршрута и взаимодействие с окружающей средой. Это полностью автономный транспорт, способный работать на любых дорогах и в любых условиях.

Раннее зарождение отрасли

Ранние 2000-е — это момент, когда формируется основа автономного вождения и проверяется в полевых условиях. Серия соревнований Управления перспективных исследовательских проектов Министерства обороны США (DARPA) сразу задала высокую планку и вынудила перейти от разрозненных подходов к системной архитектуре: многосенсорное восприятие (лидар, камеры, радары) и одометрия; калмановские фильтры для оценки состояния; простые, но предсказуемые регуляторы руления, тяги и торможения с учетом динамики машины и задержек вычислений. Двести сорок километров по пустыне, без инфраструктуры, с резкими перепадами рельефа, пылью, камнями и минимальной внешней поддержкой — ни одна машина не финишировала. Первый Grand Challenge (март 2004) по пустынному маршруту никто не завершил, что зафиксировало реальные ограничения тех лет: несогласованность каналов обмена данными, несовершенные датчики, пропадание спутникового сигнала и механические отказы. Но этот провал сформировал отправную точку для следующих стартов, например, необходимы алгоритмы слияния источников данных, дублирование датчиков, карты занятости. Карты занятости — это решетчатые карты вокруг машины, где каждая ячейка хранит вероятность занятости: занята препятствием, свободна или неизвестна в реальном времени. Это базовый рабочий слой для локального планирования. На практике карта занятости опирается на следующие компоненты:

- **Дискретизация пространства.** Задаем сетку (обычно 2D вокруг машины, 3D/воксели — для складок рельефа/мостов). Разрешение для дороги — порядка 0,1–0,25 м на ячейку; радиус — десятки метров вперед (типично 50–120 м).
- **Модели сенсоров и лучевой апдейт.** Для каждого лидара/радары/камеры считаем, что «луч» от датчика до точки попадания прошел через свободные ячейки и уперся в занятую. Эти ячейки обновляем **байесовски**: вероятность занятости выше в точке возврата, ниже вдоль луча. Камеры обычно дают маски/границы, которые переводят в «занято/свободно» через калибровку и проекцию на землю.
- **Слияние источников.** От разных датчиков в одной ячейке накапливаем доказательства (с весами под шум и дальность). Несогласованные показания не «ломают» карту: они меняют вероятность, а не перезаписывают истину.
- **Слои и «инфляция».** Поверх «сырых» занятостей формируют **стоимостную карту** для планировщика: статический слой (бордюры/стены из HD-карты), динамический слой (то, что увидели сенсоры), **инфляция** (расширение препятствий на радиус безопасного габарита + запас).

- **Время и динамика.** Ячейки «стареют»: если датчик давно не подтверждал занятость, вероятность затухает к «неизвестно». Для движущихся объектов используют отдельный трекинг и/или «динамическую занятость» (скорость/направление).
- **Роллинг-окно.** Карта хранится в подвижной системе координат машины; при движении окно «прокручивается». Для бездорожья возможна и глобальная привязка (накопление рельефа), но локальный план работает в окне.
- **Защита от артефактов.** Пыль/дождь/блики сводятся к порогам по интенсивности, временному сглаживанию апдейтов, маскам «возможного дропа» лидера, доверию к радару для дальних целей и скоростей.

В 2005 победил «Stanley» стэнфордской команды (Себастьян Трун, Майкл Монтемерло и др.). Архитектура — типичная для своего времени: несколько 2D-лидаров и камера для оценки типа покрытия, GPS, локальная карта занятости перед машиной, ограничение скорости по «шероховатости» трассы и стабильно отлаженный контур управления. Маршрут задавался файлом с опорными точками и коридором; задача машины — ехать в коридоре, избегать препятствий и не терять устойчивость при пыли и неровностях. Stanley прошел ~132 мили (~212 км) менее чем за 7 часов.

Городской «Urban Challenge» (ноябрь 2007) опять поднял планку: к автономной езде добавились правила дорожного движения, перекрестки, обгоны, парковка и взаимодействие с другими участниками. Победитель — «Boss» команды «Tartan Racing» (научные лидеры — Уильям «Ред» Уиттакер, Крис Урмсон и др.). Ключ к успеху — модульность и «иерархия поведения»: поведенческое автоматное состояние (ехать по полосе, уступить, объехать), локальный планировщик с учетом динамики препятствий и жесткая телеметрия безопасности, миссионное планирование по сети дорог RNDF — *Route Network Definition File* («файл описания сети дорог/маршрутной сети»). Глобальный маршрут по RNDF — это высокоуровневое описание маршрута, который планировщик строит по заданной «схеме дорог» соревнования DARPA Urban Challenge. В этой схеме дороги задаются в текстовом файле Route Network Definition File: он описывает доступные сегменты и полосы движения, их ширины (опционально), набор опорных точек (waypoints) в координатах, явные связи «вход-выход» между полосами на перекрестках, стоп-линии и контрольные пункты (чекпоинты). Кроме дорог RNDF задает «зоны» — площадки вроде парковок, для которых дается периметр без внутренних точек; въезды и выезды в зоны также указаны явно. Сама миссия (какие именно контрольные пункты надо посетить и в какой последовательности) и ограничения скорости на сегменты приходят из

отдельного файла Mission Data File (MDF). RNDF фиксирует топологию и геометрию, MDF — последовательность целей и скоростные лимиты.

Как из этого получается «глобальный маршрут». Система читает RNDF и MDF, строит ориентированный граф, в котором вершины — это опорные точки RNDF (включая обозначенные чекпоинты), а ребра — допустимые перемещения по полосам и через объявленные связи «вход-выход» на перекрестках и в зонах. Дальше обычный графовый поиск по стоимости: суммарная длина, штрафы за маневры, ограничения скорости из MDF и др. Результат — цепочка полос/опорных точек от текущего положения к ближайшему обязательному чекпоинту, затем к следующему и т. д. Этот маршрут «глобальный» в том смысле, что он игнорирует текущие подвижные помехи: с препятствиями и разъездами работает уже локальный планировщик на основании перцепции и локальной карты занятости. Такой разбор слоев (RNDF/MDF → граф → глобальный путь → локальное поведение) описывался в технических отчетах команд Urban Challenge и лег в основу «классического стека».

В RNDF все элементы имеют иерархические идентификаторы вида «М» (сегмент), «М.N» (полоса в сегменте), «М.N.P» (Р-я опорная точка полосы); связи на перекрестках задаются явными парами «вход→выход», стоп-знаки привязаны к конкретной точке и трактуются как стоп-линия, проходящая через нее, чекпоинты — это тоже точки, прохождение фиксируется «пересечением носом» автомобиля. В MDF для каждого сегмента/зоны задается мин/макс скорость в милях в час. Документом также допускается неполнота: местами ширина полосы может отсутствовать, плотность опорных точек — быть низкой, а какой-то участок из RNDF оказаться перекрыт; эти случаи обязан корректно обрабатывать уже онлайн-уровень системы.

Технологическим сдвигом стал промышленный 3D-лидар Velodyne HDL-64E: непрерывное облако из 64 лучей дало устойчивую геометрию окружения на десятки метров во все стороны, что упростило построение локальных карт и отслеживание объектов.

Стэнфорд занял второе место с «Junior» (Майкл Монтимерло, Себастьян Трун, Дмитрий Долгов, Ян Беккер и др.), показав аналогичный разбор задачи на уровне: глобальный маршрут по RNDF, поведенческие правила для перекрестков и зон, 3D-лидар Velodyne + дополнительные лидары для «ближнего боя», трекинг машин по лидарам и локальное планирование с учетом габаритов и кинематики. Публичные материалы «Junior» подробно описали «скелет» городского стека: от синхронизации сенсоров и оценки позы до перцепции в облаках точек и поведенческих автоматов для маневров. Дмитрий Долгов — впоследствии стал техническим директором и заместителем генерального директора Waymo.

Во всех конкурсах 2005–2007 годов повторялись одни и те же паттерны: слияние разнородных сенсоров в фильтрах состояния, локальные карты занятости/облака точек как рабочее пространство для перцепции и планирования, ранжированные правила поведения, и простые устойчивые регуляторы на выходе. Именно эта связка и получила название «классического стека» — ее позже унаследовали корпоративные программы 2010-х.

Период 2009–2016 — это переход от академического к корпоративному уровню инженерии и безопасности. Проект Google, стартовавший в январе 2009 под руководством Себастьяна Труна, через два года был публично показан, а в декабре 2016 оформлен в отдельную компанию Waymo внутри Alphabet. За этот промежуток команда отработала базовый набор практик: езда с водителями-резервистами, строгие критерии допуска маршрутов и выпуск регулярных отчетов о прогрессе. Осенью 2015 года Google провел первый проезд без человека за рулем на публичной дороге в Калифорнии.

Технически в начале корпоративной эпохи закрепляется связка многосенсорного восприятия и высокодетальных карт. Лидар и камеры дополняются радаром; одометрия и GPS стабилизируют позиционирование; локальные сетки занятости и поведенческие автоматы управляют маневрами на перекрестках и в потоке. Отдельное направление — «карты-как-датчик»: HD-карты используются не только для навигации, но и как внешний контекст для локализации и проверки гипотез восприятия. Именно в эти годы Waymo формализует часть своих процессов в открытых докладах и методических записках: как планируются тесты, как валидируется поведение и как разделяются обязанности между разработкой, испытаниями и операциями.

Отрасль ощутимо меняется под воздействием прорыва в глубоких нейросетях из общего машинного зрения. С 2012 года сверточные сети переходят из академических тестов в практическую плоскость, и уже в 2012 появляются профильные дорожные датасеты. KITTI — это общий эталон-датасет для «дорожного машинного зрения». Он дал единый набор видео и измерений, на которых разные команды могли сравнивать свои алгоритмы «в одинаковых условиях». На нем проверяли сразу ключевые умения: понимать глубину и расстояния по паре камер, оценивать сдвиг картинки между кадрами и по нему восстанавливать движение автомобиля, а также находить машины и пешеходов не только на плоскости, но и в трехмерном виде. Cityscapes сделал то же, но с акцентом на качество разметки. В нем вручную и очень подробно размечены пиксели улиц, зданий, тротуаров, разметки, людей и транспорта. Это позволило честно сравнивать и развивать алгоритмы, которые раскрашивают кар-

тинку по классам и отделяют конкретные объекты друг от друга. Параллельно работают тематические корпуса (например, Caltech Pedestrian с 2009 года) — они формируют практику воспроизводимых сравнений в задачах обнаружения пешеходов и дают индустрии общий язык метрик.

Аппаратная база на борту ускоряется и миниатюризуется. Если в середине 2000-х вычисления нередко выполняли серверной стойкой в багажнике, то к 2015 выходят специализированные автомобильные платформы GPU-класса. NVIDIA DRIVE PX (2015) — двукристальная Tegra X1 с приемом до 12 камер и производительностью свыше терафлопса для задач восприятия в реальном времени; на CES-2016 представлен DRIVE PX 2 с жидкостным охлаждением, 12 CPU-ядрами и заявленными ~8 терафлопс и «десятками TOPS» для глубинных моделей. Это переводит обработку из лаборатории в серийный форм-фактор и открывает путь к стандартизированным бортовым компьютерам (ECU) для автопилота.

Массовый автопром внедряет ассистенты водителя. К середине десятилетия Mobileye выпускает камеры с процессором EyeQ для бортового восприятия (специализированные SoC/ASIC под задачи ADAS/автопилота). В Японии в 2016 году выходит Nissan ProPILOT как серийная система помощи на автомагистрали (удержание в полосе и адаптивная скорость), что является практическим примером автопилота уровня L2. В октябре 2015 Tesla добавила в бортовую прошивку набор функции помощи водителю. Маркетинговое название набора функций было **Autopilot**, они позволяли держать полосу и адаптировать скорость на шоссе, плюс делать автоматическую парковку.

В 2016 году произошло сразу два поворотных события. Во-первых, после смертельной аварии во Флориде компания-поставщик Mobileye публично заявила о разрыве сотрудничества, пояснив это «слишком смелой эксплуатационной политикой» Tesla. Во-вторых, в октябре 2016 Tesla запустила HW2 (восьмикамерный обзор, радар, 12 ультразвуковых датчиков) на базе автомобильного компьютера NVIDIA Drive PX2 и предложила два коммерческих пакета: Enhanced Autopilot — расширенные опции автопилота и опцию FSD (Full Self-Driving — «полное самоуправление») как будущие возможности.

Следующий крупный рубеж — собственный вычислитель HW3, также называемый FSD Computer. 22 апреля 2019 Tesla на «Autonomy Day» показала плату с двумя ASIC, заявив производительность порядка сотен TOPS (триллионы операций в секунду) и перевела свежие Model S/X/3 на этот компьютер. С этого момента курс компании — полная внутренняя разработка: своя плата, свой компилятор, своя телеметрия и процесс обновлений.

С 2021 года Tesla последовательно упростила набор датчиков, обозначив курс Tesla Vision — только камеры без радара (сначала для Model 3/Y в Северной Америке, далее шире). В 2022 компания начала отказываться и от ультразвука, объявив о переходе к «vision-based occupancy network» — сети, которая по видео строит вероятностное 3D-представление занятости вокруг машины (по сути, «сетку занятости», но нейросетевым способом). Это изменило акценты: вместо «слияния разнотипных датчиков» — ставка на многоракурсное видео и крупные модели восприятия.

Архитектурно 2022–2024 годы — это движение к более цельным нейросетям на борту. На мероприятиях «AI Day» команда показывала occupancy network и переход от множества «ручных» правил к крупным моделям, которые предсказывают сцену и поведение напрямую из видеопотоков. В 2024–2025 Tesla стала распространять версию, где основной контур вождения обучается «сквозным» методом: от сырого видео камер к управляющим действиям, с переименованием FSD Beta в FSD (Supervised) — «под надзором». Юридически это по-прежнему ассистент L2, требующий постоянного внимания водителя.

После серии инцидентов NHTSA (National Highway Traffic Safety Administration — Национальная администрация безопасности дорожного движения США) провела длительное расследование и в декабре 2023 было объявлено масштабное отзывное обновление (software recall) для более чем 2 млн автомобилей: усиление контроля вовлеченности водителя и ограничений режима Autosteer. Весной 2024 NHTSA закрыла расследование EA22-002, одновременно открыв процедуру Recall Query для оценки эффективности исправлений.

Для бильярда я сознательно выбрал ту же линию, что и Tesla для своего проекта: меньше железа, больше данных. Да, я мог бы поставить вокруг бильярдного стола гроздь скоростных калиброванных камер, синхронизировать их аппаратными триггерами, раз в неделю прогонять полную перекалибровку и действительно иметь «идеальные» траектории бильярдных ударов. Но интуиция подсказала, что стоимость и операционные издержки растут быстрее точности работы. Каждая камера — это отдельная калибровка, температурный дрейф, смещения креплений, уход фокуса, грязь на стекле, рассинхронизация кадров. Чем больше сенсоров, тем больше вероятность, что одна мелочь ломает весь пайплайн, который получается на порядок сложнее. Я пошел противоположным путем: одна недорогая камера или смартфон, максимально простой код, сквозная аналитика, мощные адаптированные нейросети, обученные на корпусе данных собственной подготовки на целевом домене вместо тонкой мультисенсорной арифметики. В итоге у меня немного кода, он устойчивый, воспроизводимый и переносится между аппаратными и

серверными сборками без танцев с бубном. Сантиметровая погрешность в координатах шара на практике не влияет ни на трактовку эпизода, ни на общую статистику: динамика, проверки на правдоподобие, фильтры состояния компенсируют шум. Это очень похоже на мотивацию отказа от лишних датчиков в автомобилях: дешевле, проще обслуживать, легче масштабировать и обновлять «по воздуху» логику. Да, приходится вкладываться в устойчивость моделей и честную работу с неопределенностью, но выигрыш в себестоимости и в скорости развертывания перекрывает выгоды «идеальной» лабораторной системы, которая за пределами студии начинает требовать непропорционального внимания.

Эволюция алгоритмов автономного вождения

В середине 2010-х реально сложились два инженерных подхода. Первый — сквозной контур «видео → управление». Нейросеть учится напрямую по данным вождения и сразу выдает руление и продольное ускорение. Преимущество — минимум ручных интерфейсов и согласование всех признаков под единую цель. Ограничения — трудная верификация и сертификация, высокая требовательность к объему и разнообразию данных, сложность явного учета правил, карт и жестких ограничений безопасности.

Второй — модульный контур. Сверточные сети решают узкие задачи восприятия: детектируют транспорт и пешеходов, выделяют разметку и свободное пространство, собирают сцену. Дальше работают отдельные блоки трекинга, предсказания траекторий и планирования с явными ограничениями по физике и правилам. Плюсы — объяснимость и тестируемость по метрикам каждого узла; минусы — интеграционная сложность, накопление ошибок на стыках и необходимость аккуратно настраивать логику взаимодействия между модулями.

В итоге точка «раздела ответственности» принципиально различается. Сквозной подход берет на себя и извлечение признаков, и выбор действий одним обучаемым блоком. В модульном сеть дает структурированное представление сцены, а решения принимает планировщик с проверяемыми инвариантами. Это разные компромиссы в безопасности, валидации и темпе развития. На практике сегодня их сближают: чаще учат «сквозь» до промежуточного уровня, а планирование и правила оставляют гибридными.

Могу и тут провести прямую аналогию с моим бильярдным проектом. Часть контура у меня тоже «сквозная»: камера и детектор сразу дают заключение по факту попадания какого-либо шара в лузу без трекинга или препроцессинга. Но итоговые решения я принимаю модульно: фильтр Калмана, сопоставление треков, «правила стола» про невозможные ускорения и допуски у бортов, проверка правдоподобия столкновений. Та же двуединая логика — быстрый обучаемый фронт и строгий решатель поверх него — позволяет не усложнять чрезмерно программный алгоритм, а статистику собирать корректно. Минусы те же, что у мира автомобилей: нужен аккуратный контроль стыков и немногочисленных порогов, разрешение противоречий. Плюсы — объяснимость, переносимость между платформами и понятные зоны для улучшений, если где-то просел детектор или логика анализа эпизода.

Продолжим про автономный транспорт. Организационно 2016 год фиксирует приход крупных автоконцернов и консолидацию стартапов:

General Motors покупает Cruise Automation для ускорения городских автопилотов; в США на дороги выходит Uber с водителем-наблюдателем. Это переводит автопилот из корпоративных исследовательских центров к городским условиям. И, наконец, появляются зачатки «движка данных»: регулярные обновления прошивок, телеметрия флота, обратная связь от водителей-испытателей и собственников, выборки редких кейсов для дообучения.

В 2016 году появился термин «Fleet learning» — это способ обучать систему вождения за счет самого парка машин. Автомобили не только получают новые прошивки «по воздуху», но и сами собирают полезные фрагменты данных из реальных поездок и возвращают их в центральный контур обучения. Дальше из этих фрагментов формируют датасеты, разметку и сценарные тесты, переобучают модели восприятия и предсказания, проверяют их офлайн и выкатывают обновления обратно на парк поэтапно.

Ключевой принцип — собирать не все подряд, а именно «обучающую ценность». На борту работают легкие правила отбора: редкие маневры, спорные детекции, резкие вмешательства водителя, высокая неопределенность модели, расхождение с картой, пограничные дистанции и скорости. Машина сохраняет короткие «окна» вокруг таких событий вместе с минимальным набором телеметрии и калибровок. В разные проекты добавляют и активное «зондирование»: если модель сомневается, она помечает эпизод как кандидат на разметку. Это экономит канал связи и хранилище, но главное — концентрирует обучение на реальных слабых местах.

Обработка на сервере включает программную разметку там, где это возможно (например, восстановление геометрии по нескольким кадрам, согласование с картами, стыковка показаний разных сенсоров), и ручную разметку для сложных случаев. Затем новые модели прогоняют не только по общим метрикам, но и по библиотеке «острых» сценариев: пешеход, закрытый машиной; мигающий светофор; мокрая разметка; ночной дождь. Перед широкой раскаткой обновления идут в «теневом режиме» на части парка: модель считает, но не управляет, а ее ответы сравнивают с действующим стеком и фактом вмешательств водителя. Если регрессов нет, долю парка и регионов расширяют.

Главная опасность регрессов в автопилоте состоит в том, что новая версия может действительно улучшить поведение в специально отобранных трудных сценариях, но незаметно ухудшить уже давно освоенные и более массовые режимы. Модель, которая стала лучше замечать пешехода из-за припаркованной машины, может начать чаще «нервничать» на тенях у обочины; обновлённый планировщик, аккуратнее про-

ходящий сложный перекрёсток, может хуже выбирать дистанцию в простом плотном потоке. Поэтому каждое изменение оценивают не только по среднему росту общей метрики, а как поиск скрытых потерь по отдельным срезам: погоде, времени суток, типам дорог, скорости движения, редким конфликтным ситуациям. В этом смысле регресс — это не просто «новая версия чуть хуже старой», а нарушение ранее достигнутого уровня надёжности в конкретном участке эксплуатационной области. Именно поэтому зрелые команды строят библиотеку сценариев, теневые прогоны и процедуры допуска обновлений так, чтобы подтвердить не только наличие прогресса, но и отсутствие деградации в уже закрытых рисках; до широкой раскатки важно доказать, что система стала лучше не ценой потери устойчивости там, где ей уже доверяли.

Здесь стоит упомянуть, что мой проект PerformStars тоже использует подход «флот-обучения» в миниатюре: мои машины — это столы, камеры, игроки и партии. Я сознательно не храню на сервере весь видеопоток, а отбираю фрагменты с обучающей ценностью. В первую очередь — сложные удары, где траектория неоднозначна: быстрые пересечения траекторий, касания с микропаузой у борта, «зависания» на кромке лузы. Дальше — редкие условия: нестандартные расцветки шаров и сукна, уличное солнце с жесткими бликами, «теплый» свет в бильярдной, нетипичные ракурсы установки камеры. Такие записи статистически выпадают из «среднего» уровня точности и требуют внимания: я помечаю их специальными тегами и отправляю в дообучение.

Технически сбор устроен просто и надёжно. Я фиксирую короткие окна видео по событию — несколько сотен миллисекунд до удара и одна–две секунды после — вместе с минимальным набором метаданных: частота кадров, «гомография» стола, размер видео, размер стола. Для разметки использую полуавтомат: базовый трекер даёт черновую траекторию, дальше физические ограничения «приглаживают» её — постоянное ускорение между столкновениями, упругий зеркальный отскок от борта. По *re-ID* я сознательно выбираю отрицательные примеры — почти одинаковые по тону шары в контакте — и добавляю к ним искусственные сдвиги баланса белого, зерно, лёгкий блюр, чтобы эмбединги меньше зависели от света и матрицы камеры. В цветовых расхождениях помогает простая нормализация по эталонным патчам сукна и белого шара; это убирает половину разлётов уже на препроцессинге.

Чтобы не обманывать себя, я удерживаю «золотой набор» эталонных эпизодов и жестко разделяю источники на обучение и проверку по столам и камерам, а не по кадрам — утечки между доменами здесь особенно неприятны. Новые веса сначала гоняются в «теневом режиме»: модель считает, но решения не влияют на итог, я смотрю на мет-

рики — потерянные треки после столкновений, стабильность идентичности при перекрытиях, точность определения факта забитого шара относительно события «исчезновения» — и только потом выпускаю обновление в работу. Если где-то метрика просела, я возвращаюсь к тем же тегам и подбираю примеры адресно, как это делают большие «флоты» на дорогах.

Отечественная школа автономного вождения

Если смотреть именно на «ранние» шаги автономного вождения в России, то до середины 2010-х речь шла скорее о базовых заделах и точечных демонстрациях, а не о крупных дорожных программах. Самый заметный инфраструктурный фундамент — восстановление и модернизация ГЛОНАСС: после провала 1990-х государство в 2001–2011 годах фактически заново довело группировку до глобальной работоспособности. Это дало промышленности стабильную спутниковую навигацию для локализации и синхронизации, без которой городские эксперименты автономного движения малоосуществимы. Практический поворот к собственно автономному вождению начинается в стране в 2015–2017 годах. Тяжелый транспорт первым получил системные пилоты: КАМАЗ и Cognitive Technologies объявили программу беспилотного грузовика, запустили полигон и полевые испытания — это была одна из первых публичных индустриальных инициатив по наземной автономке в РФ. В этот же период обсуждались специальные тестовые «города» и режимы колонного движения, что отражало во многом «классический стек» (лидары, камеры, GPS/ГЛОНАСС/ИНС, сетки занятости, локальные планировщики).

Городской сегмент практически одновременно пошел двумя линиями. Исследовательские институты и вузы показали собственные прототипы: НИЦ НАМИ представил электрический беспилотный микроавтобус-концепт, а МАДИ (Московская административная дорожная инспекция) запустила проект «Караван» и вывела университетские машины на закрытые и затем на открытые участки дорог для отработки инфраструктурного взаимодействия и маршрутов. Это были первые «уровневые» демонстрации городского движения в кооперации вузов, городских структур и регуляторов.

Параллельно коммерческий вектор задал Яндекс: в 2016 компания сформировала команду, в 2017 показала первый прототип, а к зиме 2017–2018 начала публичные заезды, в том числе в снег — с уже «корпоративным» стеком сенсоров (лидары, радары, камеры) и серийными GPU-платформами на борту. Это был быстрый переход от лаборатории к уличной эксплуатации с северными сценариями, что сильно подняло планку требований к восприятию и планированию в условиях зимы.

Через год в Иннополисе стартовал роботакси-сервис: сначала с инженерами-наблюдателями на пассажирских местах, затем с постепенными экспериментами без человека в салоне и дистанционным контролем безопасности. Эта линия — от «снега» к «перевозке пассажиров» — как раз и показывает переход из лаборатории в городскую эксплуатацию. Параллельно шли международные демонстрации (включая CES в Лас-Вегасе) и партнерство с Hyundai Mobis по системам уровня 4–5, а

внутри самой платформы Яндекс сделала, пожалуй, самый показательный шаг — разработала собственные лидары с «программно определяемым» шаблоном сканирования и заявленным дальним боем до ~500 м, чтобы управлять главным сенсором «под задачу», экономя время и себестоимость и получая доступ к «сырым» данным. На дистанции этот стек устоялся: после корпоративной реструктуризации команда продолжила путь в виде Avride, уже с программами роботакси и доставочных роботов в США и новой волной сотрудничества с Hyundai. Это редкий для отрасли пример, когда технологическая школа, выросшая на российских дорогах, органично продолжает масштабирование на другом рынке.

С этого момента отрасль резко ускорила — от пилотов к регулярным испытаниям, а затем и к прикладным решениям в смежных доменах (железная дорога, агроавтономка), крупным индустриальным программам на дорогах общего пользования, прикладным решениям для «тяжелых» отраслей и инфраструктурным технологиям компьютерного зрения.

К 2020 система Cognitive Agro Pilot провела полноценные уборочные кампании на комбайнах: автономная навигация по неровным, пыльным полям с непредсказуемыми препятствиями — отличный тренажер для устойчивых детекторов, сегментации рядов и траекторного планирования. Та же школа доведена до железнодорожной сферы — испытания ассистента машиниста с распознаванием светофоров, стрелок, препятствий и сценариев проезда сигналов.

Сегмент «городского транспорта» в России также развивался через совместные проекты автопроизводителей и институтов. КамАЗ вместе с НАМИ (Центральный научно-исследовательский автомобильный и автомоторный институт) и технологическими партнерами прошел путь от автономного грузовика-демонстратора до электрического шаттла, который показывали на федеральных площадках; существенно, что испытания планировались и на магистрали М-11, ради чего готовили специальную цифровую инфраструктуру. Важно не путать выставочные прогоны с регулярной эксплуатацией, но инженерная наработка — сенсоры, карты, каналы связи — формировала компетенции и внутри автопрома, и у системных интеграторов. Отдельной линией идет корпоративный отдел: SberAutoTech в 2021 представил концепт собственного автономного «капсульного» транспорта FLIP.

Развивается инфраструктурное компьютерное зрение на уровне города: идентификация, переидентификация, сопоставление траекторий людей и автомобилей в огромных видеопотоках. Здесь российские команды сделали вещи, которые влияют на автономное вождение опосредованно, но существенно: это и устойчивые эмбединги для ge-ID, и

массово работающие пайплайны с миллионами событий в сутки. Публично наиболее заметны VisionLabs и NtechLab: первая известна, среди прочего, запуском оплат «по лицу» в Московском метро и экспортом этой технологии, вторая — масштабными внедрениями ge-ID в городских системах видеонаблюдения. При этом вокруг них закономерно шла дискуссия об этике и применимости технологий распознавания в общественных системах — важный контекст для любой инфраструктуры.

Параллель с развитием российского машинного зрения можно провести и относительно моего проекта: я решал переидентификацию игроков у бильярдного стола как прикладную задачу компьютерного зрения с жесткими доменными ограничениями. В отличие от городских потоков у меня заранее известен круг участников (обычно двое), фиксирована сцена, а события — удары и переходы хода — заданы регламентом. Практический эффект — автоматическая привязка каждого удара к правильному игроку и, как следствие, полноценный онлайн-скоринг: серии, фолы, время на удар, тепловые карты стола, «темпы» партии. Такой ge-ID становится базовым модулем продуктовой линейки: локальная установка в клубе, облачный сервис для трансляций с оверлеями, пакет функций для интеграции в сторонние приложения подсчета очков. Порог входа низкий — работает на типовой камере и не требует сложной инфраструктуры, поэтому это решение масштабируемо: бильярдные клубы, любительские лиги, федерации. География и экспорт здесь естественны: правила игры универсальны, а мой стек не привязан к конкретным брендам камер или освещению.

Есть и академический вклад, который часто остается «за кадром». Открытый симулятор CARLA, Car Learning to Act («автомобиль, обучающийся действовать»), сделанный в 2017 году в команде с участием Алексея Досовицкого и Владлена Колтуна, стал стандартной лабораторией для проверки стеков восприятия и планирования: на CARLA воспитывались и классические модульные пайплайны, и имитационное обучение, и дальнейшие смеси с усилением. Такие платформы экономят месяцы полевых испытаний, а самое главное — дают воспроизводимые «сценарии правды». Вклад симулятора в мировую повестку автономного вождения трудно переоценить: его цитируют в учебных курсах и используют все — от университетских команд до промышленных исследовательских лабораторий.

Российские команды хорошо проявляли себя в доведении машинного зрения до уровня промышленной эксплуатации: снег, грязь, сельское хозяйство, сложные городские сцены и большие нагрузки на инфраструктуру. Отсюда — ставка на цельные технологические стеки и надежность: собственные сенсоры, плотная интеграция восприятия с картами и связью. И отсюда же — органичный выход за пределы страны: когда

стек оттестирован в тяжелых условиях, он конкурентоспособен и на других рынках. Именно такое сочетание и двигает отрасль: машинное зрение становится ремеслом, способным работать не только в солнечной Калифорнии, но и в февральской Казани, на трассе М-11 и на поле с влажной кукурузой.

Рождение эры автономного транспорта

В декабре 2018 Waymo переводит роботакси в сервисный режим: запущен Waymo One в Финиксе (Аризона), а в октябре 2020 начаты полностью «безводительские» поездки для широкой публики в той же зоне обслуживания. И далее с 2020-х она стабильно расширяет беспилотные зоны и тарифные разрешения в Финиксе, Сан-Франциско и Лос-Анджелесе: в марте 2024 года калифорнийский регулятор разрешил коммерческие поездки без водителя сразу в Лос-Анджелесе и на полуострове Сан-Франциско.

В Китае к 2022–2023 годам Baidu Apollo, Pony.ai и AutoX получили первые полномасштабные допуски на полностью «безводительские» поездки в Пекине, Шэньчжэне, Гуанчжоу и Шанхае — с реальными коммерческими режимами и без водителя в салоне.



Беспилотный автобус на дороге в центре Гуанджоу (Китай)

Эти системы используют «тяжелую» сенсорику (лидар+камеры+радары), точные HD-карты и строгие процедуры безопасности: область применения ограничена геофонами, но поведение в них отлажено и сертифицировано.

К 2020м были алгоритмически закреплены в продакшене трансформеры и представление сцены в виде проекции «вид сверху» (BEV, bird's-eye view), которые стали стандартом для объединения камер, лидара и радара. Семейства нейросетей BEVFormer/BEVFusion задали канонический способ склейки многовидовых изображений и 3D-точек в единой карте занятости; идея занятых карт (сеток) дополнила детекцию там, где геометрия и видимость неполны. Для лидара это зрелые BEV-детекторы (MV3D, затем быстрые PointPillars в CVPR-2019 с 2D-свертками по псевдокартам), для камер — прямые лифты в BEV из многокамерных изображений (Lift-Splat-Shoot, ECCV-2020). Такой формат удобен для последующих модулей трекинга и планирования, поскольку дает единое метрическое поле. Трекинг объединяется с предсказанием траекторий и оценкой неоднозначности будущего. Появляются многомодальные предсказатели, выдающие набор альтернативных путей с вероятностями: MultiPath (Google/Waymo, 2019), CoverNet (Lyft, 2019), Trajectron++ (2019–2020). Для «векторизованных» карт и взаимодействий агентов предлагаются модели уровня VectorNet и LaneGCN (2020). На практике такие блоки заменяют жесткие эвристики перекрестков статистическими оценками и дают входы планировщику. В прогнозировании движения утвердились многомодальные предсказания с вероятностями и совместное моделирование агентов: от графовых моделей VectorNet до сценовых трансформеров Waymo. Под это вышли и крупные открытые датасеты движения (Waymo Open Motion Dataset, Argoverse Forecasting, Lyft Level 5). Датасеты переходят из «одноканальных» в многосенсорные со стабильной оценкой качества. После KITTI/Cityscapes появляются nuScenes (2019/2020) с 6 камерами, 5 радаром и 1 лидаром по кругу; Lyft Level 5 (2019) — многосенсорный городской корпус; Waymo Open Dataset (2019) — крупноформатный набор с лидарами и многокамерными кольцами, что резко ускоряет воспроизводимые сравнения по 2D/3D-детекции по трекингу и предсказанию. Следом выходят Argoverse (2019) и ежегодные челленджи. Симуляция становится обязательной частью цикла валидации. Waymo публично рассказывает о собственной среде Carcraft. Академическое сообщество получает открытые симуляторы CARLA (2017) и Microsoft AirSim (2017) — стандартные стенды для отработки восприятия, планирования и регрессий.

Отрасль уже не живет гонкой нейросетевых архитектур. Успех определяется объемом и качеством флотских и симуляционных сценариев, прозрачностью метрик, глубиной обучения на борту и в «облаке», скоростью обновления прошивки «дорожные случаи → дообучение → регресс-тест → выпуск обновления». Геофенс-подход и камероцентричный подход сосуществуют и обмениваются практиками: первый оттачивает гарантии и операцию, второй ускоряет генерализацию и стоимость вывода. На горизонте — дальнейшая унификация по слиянию данных в

виде сверху и трансформерам, более тесные связки восприятия и планирования, и правила безопасности прямо внутри обучаемых систем.

Аппаратная база смещается к унифицированным автомобильным вычислителям с запасом под «большие» модели: семейство NVIDIA DRIVE Orin стало де-факто платформой для серийных продвинутых ассистентов вождения и автономных систем, а DRIVE Thor, анонсированный для авто 2025 года, ориентирован на трансформеры/генеративные задачи и конвергенцию «восприятие-прогноз-планирование» на одном кристалле. Это снижает инженерную сложность интеграции и упрощает соответствие стандартам безопасности.

В основании систем автономного вождения лежат сенсоры и синхронизация: многокамерные кольца с известной внутренней калибровкой, лидары с плотностью лучей и стабильной температурной геометрией, радары для дальних скоростей и работы в непогоду, инерциальные модули с гироскопами и акселерометрами, а также спутниковое позиционирование. Все датчики жестко привязаны ко времени и друг к другу: бортовые контроллеры выставляют метки времени, позиции антенн и центров проекции перечислены в единой системе координат автомобиля. Уже здесь закладываются простые проверяемые ограничения: частоты кадров, задержки, динамический диапазон камер, маски бликов и загрязнений, допуски на дрейф.

Над сенсорным слоем работает локализация и карты. В потоке — визуальная или лидарная одометрия, слияние с ИНС (инерциальной навигационной системой) через фильтры состояния, корректура при наличии спутников, распознавание ориентиров для привязки к окружению. Карты высокого разрешения используются как приоритет, а не единственный источник: по ним уточняют полосу и геометрию перекрестка, но при расхождении с фактическими, веса снижаются и стек переключается на понимание сцены через бортовые камеры в реальном времени. Слои карты разделены по типу данных — статическая геометрия и семантика, ограничения скорости, расположение светофоров и знаков, зоны особого режима.

Далее идет восприятие в представлении «вид сверху». Многокамерные изображения и облака точек лидара приводятся к единому формату в плоскости дороги; получается BEV-карта с признаками занятости: свободно, занято статическим объектом, занято динамическим объектом с оценкой скорости и направления. Здесь же предсказывается разметка, контуры островков, бордюры, пересечения и проезжие коридоры. Для камер используются модели, которые восстанавливают глубину и семантику и «поднимают» их в BEV; для лидара и радара — преобразования к псевдокартам и нейросетевые сверточные/трансформерные фьюжн-модели. Детекторы и сегментаторы дают объекты и контуры на

изображении. Все вероятности калибруются, чтобы численные «уверенности» соответствовали реальной частоте ошибок; отдельно учитывается «модельная» неопределенность самой сети.

Поверх восприятия работает трекинг и прогноз. Состояние объектов описывается в метрических координатах, с оценкой скоростей, габаритов и ковариаций; ассоциация между кадрами учитывает динамику и геометрию, а в сложных сценах опирается на вероятностные гипотезы соответствия. Прогноз выдает не один путь, а набор альтернатив с вероятностями и временами до конфликтов; взаимодействие участников моделируется графовыми или сценовыми моделями, чтобы в узких местах не возникали противоречивые предсказания.

Решения формирует связка планирования поведения и планирования траектории. На первом уровне система выбирает допустимый маневр в текущем контексте — стоять, ехать прямо, уступить, перестроиться, повернуть на заданном пересечении — опираясь на правила дорожного движения и формальные ограничения безопасности. На втором — ищет динамически выполнимую траекторию в коридоре свободного пространства, учитывая габариты, комфорт, сцепление и запас до препятствий. Распространена гибридная схема: политика поведения может быть частично обучаемой, но поверх нее действуют жесткие условия — ограничения по расстояниям и тормозным путям, барьерные функции и гарантии «не столкнуться при любых ошибках в пределах модели». Для непосредственного управления используются регуляторы, обычно предиктивные с ограничениями на углы и скорости, а также проверяемые стратегии аварийного торможения и стабилизации.

Поперечные слои поддерживают всю систему. Диагностика отслеживает здоровье сенсоров, задержки, насыщение вычислений и запас по температуре; деградация определяет, какие функции отключать и как изменять поведение при отказах. Кибербезопасность реализуется через защищенные коммуникации, разделение зон доверия и выполнение требований стандарта по автомобильной информационной безопасности; телеметрия собирает короткие окна событий, а «фабрика данных» формирует датасеты, поддерживает программную и ручную разметку, проводит регрессионные тесты и управляет выпуском версий. Симуляция дополняет реальный мир сценариями редких событий, интегрируется с картами и позволяет повторять инциденты с вариациями параметров до устранения причины. Обновления разворачиваются поэтапно, с «теневым режимом», когда новая модель считает, но не влияет на управление, а ее ответы сравниваются с действующей системой и фактами вмешательств.

Если назвать используемые семейства методов без углубления, то в восприятии это детекторы и сегментаторы для камер и лидара,

трансформерные и сверточные фьюжн-модели для объединения камер-лидара-радара, а также сетки занятости как универсальный слой для планирования. В прогнозе — многомодальные модели траекторий с вероятностями и совместное моделирование сцены с помощью графов и сценовых трансформеров. В планировании — машины состояний для поведения и оптимизационные методы для траекторий, где предиктивное управление встроено в ограничения по безопасности и динамике шасси. Параллельно развиваются сквозные головы, которые учатся от сенсоров до промежуточных представлений или даже до управляющих воздействий, и мировые модели — подход к обучению на длинных горизонтах с имитацией динамики сцены. Их рассматривают как направление развития, но выпускают в эксплуатацию через те же процессы валидации, калибровки и контроля регрессий, что и модульный стек. Именно эта связность пластов и дисциплина данных позволяют переносить решения между городами и сезонами, не меняя базовую архитектуру.

ЧАСТЬ 5

Видео-понимание: видеть не кадр, а историю

С изображениями все просто: мы ловим мгновение и пытаемся назвать, что на нем. Видео заставляет думать иначе. Важен не один кадр, а то, как они связаны. Мы хотим распознать действие — «передача мяча», «падение», «застревание ленты» — и понять, когда оно началось и когда закончилось. Иногда задача шире: составить сжатое описание того, что происходит в клипе, превратить минуты в несколько осмысленных фраз. Бывает наоборот: нас интересуют аномалии, которые никто заранее не собирал в выборку, — редкие сбои, нарушающие привычный ритм.

Разметка в таких задачах отражает эту «временную» природу. Для распознавания действий метят целиком клипы: короткие отрезки видео скормливают модели как примеры «передачи», «ударов», «подъема коробки». Когда важны границы событий, аннотации становятся интервальными: у каждого действия есть начало и конец в формате времени или номеров кадров. Для описаний добавляют текст: одна или несколько фраз, которые, по мнению человека, лучшим образом передают смысл увиденного. Реальность почти всегда приносит шум: у разных разметчиков границы будут расходиться, а «подачу» и «передачу» кто-то запишет как одно и то же. Это не ошибка — это свойство мира, и модель приходится учить жить с такой неопределенностью.

В безопасности задача может звучать просто: «заметить падение и позвать персонал». Но за простотой скрывается длинный контекст. Человек тянется за вещью, теряет равновесие, пытается удержаться и падает — все это несколько секунд, а иногда и десятки. Если смотреть только на мгновения, легко спутать движение руки за телефоном с падением. Производство предъявляет свои требования: нам нужен сигнал «застряла лента» или «человек вошел в опасную зону». Камера видит повторяющийся узор: коробка за коробкой, все одинаково — и вдруг цепочка нарушается. Ключ — именно нарушение ритма, не один неудачный кадр. В спорте важны и типы действий, и их границы: пас, прием, удар, офсайд. Тут модель работает как «счетовод времени»: она должна не только представить кадрам «этикетки», но и собрать их в цельные, не пересекающиеся эпизоды. В медиа-поиске логика другая: мы хотим находить по тексту — «человек открывает красную дверь» — и возвращать подходящие фрагменты видео. Тогда на первый план выходит соответствие слов и образов, а не только чистая динамика.

Метрики здесь нужно делать адекватные особенностям задачи. Когда мы классифицируем клип, удобно смотреть, насколько часто «первый ответ» модели совпадает с истиной; если нас устраивает, что правильный класс входит хотя бы в пятерку лучших гипотез, — то и это можно учитывать. Для поиска и локализации событий нужна точность по времени. Используется простая идея пересечения и объединения, но уже не по площадям, а по отрезкам времени: чем больше доля пересечения предсказанного интервала с истинным, тем лучше. Если пересечение превышает порог, засчитываем попадание, а дальше усредняем качество по множеству порогов строгости — так меньше соблазна «подгонять» систему под один-единственный критерий. С текстовыми описаниями ситуация тоньше: автоматически сравнивать фразы — все равно что спорить о стиле. BLEU (Bilingual Evaluation Understudy) — алгоритм оценки качества текста, переведенного машинным способом с одного естественного языка на другой. Он сравнивает кандидатный перевод с одной или несколькими эталонными версиями, вычисляя совпадения n-грамм и учитывая штраф за краткость.

CIDEr (Consensus-based Image Description Evaluation) — метрика оценки качества генерируемых текстовых описаний изображений. Она измеряет сходство сгенерированного предложения с набором эталонных предложений, созданных людьми, используя векторизацию на основе TF-IDF для повышения согласия с человеческими оценками. TF-IDF — это способ присвоить словам вес по их «полезности» для отличия одного текста от другого. Вес каждого термина складывается из двух множителей. Первый — term frequency, или TF: насколько часто слово встречается в данном документе. Второй — inverse document frequency, или IDF: насколько редким это слово является во всей коллекции документов. Идея проста: слова вроде «и», «это», «на» появляются почти везде и мало что говорят о содержании, поэтому их вес должен быть маленьким; термины вроде «спектрограмма» или «пирамида» встречаются реже и сильнее «подписывают» тему — их вес должен быть больше.

BLEU применяется в видео-капTIONинге для оценки n-граммного совпадения с оригинальными описаниями, но действительно чувствителен к вариациям формулировок. CIDEr, изначально разработанный для изображений, адаптирован для видео и фокусируется на консенсусе с человеческими описаниями через TF-IDF-векторизацию, что делает его подходящим для оценки качества описаний. Дополнительные проверки, такие как retrieval-метрики и человеческие оценки, стандартны для компенсации слабостей BLEU и CIDEr в реальных сценариях. Метрики вроде BLEU и CIDEr полезны для грубой прикидки на больших массивах, но сильно страдают от синонимов и парафраз. На практике к ним добавляют более «прикладные» проверки — например, насколько хорошо по

подписанному клипу находится исходное видео среди других, — и выборочные людские оценки.

Модели для видео строятся вокруг идеи, что время — такой же носитель смысла, как и пространство. Один подход делает время «третьим измерением»: вместо того чтобы обрабатывать кадры по отдельности, сеть учится извлекать признаки сразу из объема — небольшой пачки кадров подряд (3D-свертки вроде I3D). Другой подход делит ответственность: одна ветвь медленно, но глубоко вникает в форму и контекст на кадрах, другая быстро улавливает движения, как бы записывая «почерк скорости» (SlowFast). Третий подход опирается на механизм внимания: он рассматривает фрагменты видео как последовательность токенов в пространстве и времени и учится выделять важные связи между ними (видеотрансформеры, например TimeSformer). Все чаще в это добавляют текст: модели учатся «сшивать» машинное зрение и язык, чтобы понимать не просто движение, а смысл сцены в словах. Практический рецепт почти всегда включает предварительную подготовку на гигантских массивах немеченого видео — модель учится предсказывать пропущенные части, согласовывать соседние кадры, сопоставлять клипы и подписи. Такая «самообученная» закалка потом резко помогает на реальных задачах с ограниченными и шумными аннотациями.

I3D представляет "inflated" 3D-свертки, расширяющие 2D-модели для захвата пространственно-временных признаков из последовательностей кадров. SlowFast разделяет обработку на медленный путь для семантики и быстрый для динамики, что эффективно для распознавания действий. TimeSformer использует трансформеры для пространственно-временного внимания, избегая сверток и фокусируясь на связях между фрагментами.

Главные трудности это длинный контекст: действие растянуто во времени, и его нельзя распознать по одному кадру. Система должна уверенно держать в памяти сотни миллисекунд, а иногда — десятки секунд. Если пытаться «впихнуть» все в один прогон, мы упираемся в память и задержку. Инженерный выход — работать кусками: брать скользящее окно кадров, аккуратно обновлять оценки по мере поступления новой информации и «договариваться» с оператором о задержке, приемлемой для приложения. Во-вторых, редкость событий. Целевые функции, которые мы пытаемся найти, например «падение человека» или «сбои на линии» — случаются редко, и модель быстро учится «узнавать» норму, игнорируя то, что мы хотим ловить. Помогает имитация редких случаев, негативные примеры (похоже, но не то), а также обучение на «аномалиях без имен»: модель изучает нормальный ритм и сигналит, когда статистика «ломается». В-третьих, шумные метки. Два разметчика поставят разные границы события и по-разному подпишут сложные эпизоды. Тут

надо усреднять разметку, обучать так, чтобы модель не наказывалась слишком сильно за небольшие сдвиги по времени и формулировке.

Еще одна ловушка — зависимость от фона. Модель «узнает» не действие, а сцену: «плавание» — это бассейн, «ездить на велосипеде» — это дорога с определенными цветами. Стоит велосипедисту проехать по песку, и ответ внезапно меняется. Избавиться от этого можно только тренировкой на разнообразии: показывать действие в разных декорациях, нарушать привычные связи аугментациями (менять фон, цветовую гамму), добавлять «жесткие отрицательные» примеры, где фон тот же, а действия нет. Камера тоже может мешать: панорама или зум выглядят как движение в кадре, хотя двигается не объект, а сама оптика. Простой способ не попасться — не делать ставку на «движение как таковое», а учиться на сочетании формы и динамики, а там, где возможно, компенсировать движение камеры по соседним кадрам.

Рассмотрим, как это складывается в двух реальных системах. На заводе нужно обнаруживать «срыв» — момент, когда коробка на ленте поворачивается и клинит конвейер. Камера смотрит сверху, кадры похожи как один: одинаковые коробки, одинаковые промежутки. В обучении мы бережно собираем редкие случаи сбоев, а затем настраиваем систему так, чтобы она «слушала» ритм: периодичность появления коробок, стабильность их ориентации, плавность перемещения. Строится окно: каждые полсекунды мы обновляем оценку «нормальности», а событие объявляем только если нарушен не один кадр, а устойчивый кусок времени. Сигнал уходит не сразу «в аварийную кнопку», а сначала в визуальное подтверждение оператору: два-три кадра с подсветкой места, где произошло нарушение. Это снижает ложные остановки и дает людям уверенность, что система не «пугается» каждого блика.

В спорте задача обратная: выделить ключевые моменты матча — «удар в створ», «опасная атака», «фол». Здесь разношерстный фон, шумная съемка, разные стадионы. Мы учим модель не только по видео, но и по комментариям матча: «гол!» или «удар мимо ворот» — слабые, но полезные подсказки. На инференсе применяем скользящий буфер в несколько секунд и сглаживаем решения во времени, чтобы «опасная атака» не мигала через кадр. Границы эпизодов уточняем постфактум: когда событие уже зафиксировано, модель «дочитывает» еще секунду-другую назад и вперед, чтобы аккуратно отрезать эпизод и подготовить клип для обзора. Качество проверяем не только цифрами от нашей модели, но и по действиям оператора: сколько моментов редактор правит руками и сколько принимает без изменений.

В медиа-поиске задачей является диалог с видео. Мы хотим, чтобы фраза «человек открывает красную дверь» находила именно такие сцены, а не любые проходы по коридору. Модель предварительно

учат связывать короткие видеоклипы и подписи к ним, чтобы близость в пространстве признаков отражала смысловое совпадение. На практике мы еще и аккуратно подбираем, сколько времени «нужно на действие»: для «открывает дверь» это секунды, а для «стоит в очереди» — минуты. Поэтому система возвращает не один точный кадр, а набор отрезков разной длины, ранжированных по уверенности, и дает пользователю быстрый просмотр — так меньше разочарований от «правильных, но бессмысленных» попаданий.

Все эти истории объединяет одно: видео — это не просто «больше пикселей», а другое мышление. Мы храним и обрабатываем время, учимся жить с редкими событиями и шумной правдой, осознанно вводим задержки и буферы, чтобы принимать решения не по отдельным мгновениям, а по устойчивым временным паттернам. Математика опирается на усреднения и сглаживания, аккуратные окна, меры совпадения по времени, внимание к связям между фрагментами. Когда система приходит в действие, она видит не просто кадры, а события — с началом, серединой и концом. Это экономит колоссальное количество времени и человеческого внимания.

Видео-свертки и понимание движения: как «слышать» время в кадре

Классические сети для картинок «видят» только один лист. Видео-свертка добавляет третье измерение и учит модель работать сразу с пачкой кадров, как если бы вы держали в руках целую стопку и рассматривали объем. Вместо фильтра, который «ездит» по высоте и ширине, мы двигаем фильтр еще и по времени — по нескольким соседним кадрам разом (ядро $t \times h \times w$).

Из такого простого расширения возникает мощная идея: движение появляется «само собой». Разницу между положениями руки на соседних кадрах свертка видит как характерный паттерн, и ей уже не нужно отдельно считать «оптический поток». Но за эту память приходится платить: обработка объема дороже, чем одного кадра, и встают практические вопросы — сколько кадров брать, с каким шагом, какой длины «временной слух» нам нужен. Если действие короткое, как удар по мячу, хватит пары десятков кадров; если медленное, как подъем коробки со стола, нужно окно длиннее и более редкая выборка кадров, чтобы не утонуть в вычислениях.

Исторически первые удачные видео-сети рождались из уже знакомых архитектур для изображений. Берут готовую, обученную на миллионах картинок, и «надувают» ее фильтры в третьем измерении, растягивая по времени (подход, известный как I3D). Это позволяет сохранить богатые зрительные знания — формы, текстуры, привычные сочетания — и добавить чувствительность ко времени. На практике это сильно ускоряет обучение и снижает требования к разметке: нейросетка уже «знает», что такое шар или рука, ее нужно научить, как они двигаются.

Чуть позже появилось наблюдение, что не все пиксели одинаково «быстрые». Например, в бильярде есть медленная, содержательная часть — очертания игроков, контуры игрового стола, лузы. Она меняется относительно плавно. Есть быстрая — движение шаров, взмахи рук. В архитектуре SlowFast это воплощено буквально: один поток смотрит видео как бы «медленными глазами», реже подбирая кадры, и отвечает за «что это в целом», а второй — «быстрыми глазами», берет кадры чаще и ловит движение. Потоки обмениваются подсказками: быстрый помогает замедленному различать похожие действия («поднял руку» — это подготовка или удар?), а медленный дает контекст, чтобы быстрый не путал шум камеры с реальным движением. Такая «двухтемповость» оказывается особенно полезной там, где важны и форма, и движение — в спорте, на производстве, в безопасности.

Параллельно со свертками в игру вступает внимание. Если разбить кадры на небольшие «плитки» и рассматривать их как элементы последовательности, можно научить модель выделять связи «что с чем связано» во времени и в пространстве (видеотрансформеры — например, TimeSformer). Плюс такого взгляда в гибкости: можно «смотреть» далеко по времени, можно уделять больше внимания рукам игрока и меньше — неподвижной трибуне. Минус — стоимость: полный просмотр связей между всеми «плитками» растет квадратично, и инженерам приходится хитрить — сначала разбирать пространство в каждом кадре, потом время между кадрами; или ограничивать внимание локальными окнами; или уменьшать «плотность» представления. Эти ухищрения позволяют достать длинный контекст там, где свертки становятся слишком «забывчивыми»: например, чтобы понять многоходовую партию или «нарастающую» аварийную ситуацию, одной секунды бывает мало.

Как это выглядит в живых задачах. Возьмем мой проект в бильярдном спорте. Короткий фрагмент — подготовка и удар по шару. Простой 3D-классификатор (на котором «надуты» 2D-веса) прекрасно отличает удар от простого переката рукой: шар описывает дугу, и даже без «пикселей скорости» паттерн узнаваем. Но чтобы отыскать в часовом видео именно этот момент и правильно отрезать его по границам, полезно добавить «двухтемповый» взгляд: быстрый путь надежен в обнаружении самого удара, медленный — в понимании, где эпизод начинается и где заканчивается. Если поверх этого еще и нужен поиск по словам — «разбой в очередной партии» — на сцену выходит мультимодальная часть: мы заранее учим систему связывать фрагменты с короткими текстовыми подписями, и тогда простая фраза вытягивает нужные фрагменты без ручной разметки.

Детекция и сегментация в видео почти всегда строятся поверх таких временных представлений. Самая надежная связка — кадр-за-кадром ищем объекты знакомым детектором, а затем «шьем» рамки во времени в траектории. Но есть способ выгоднее использовать прошлое. Если у нас уже есть маска человека в предыдущем кадре, можно «протащить» ее вперед, примерно оценив, как сместилась картинка, и попросить сеть не рисовать маску с нуля, а уточнить проташенную (так экономится и время, и устойчивость границ). Похожим образом работают сегментаторы действий во времени: сначала появляется грубая карта «где что происходит», а потом, когда накапливается контекст, границы эпизодов подрезаются точнее. И здесь, и там работает одна мысль: не забывайте, что время — это не шум, а источник достоверных данных, и рациональнее переиспользовать то, что уже посчитано, чем считать заново каждый кадр.

Работа в реальном времени добавляет свои особенности. Видео от камер приходит не равномерно: кадры могут опаздывать, местами «падать», метки времени слегка гуляют, невозможна перемотка кадров назад и вперед во времени. Если просто «ждать, пока окно заполнится», задержка увеличится, и система может стать «рваной». На практике помогают небольшие буферы с ограничением или без. Хорошая видеосистема умеет контролируемо деградировать: снизить разрешение, реже брать кадры, отложить тяжелую часть модели, но при этом продолжать давать стабильные ответы на базовом уровне.

Где здесь место трансформеров и мультимодальности, помимо уже упомянутого поиска по словам? Когда нужен очень длинный контекст — рекламный ролик, новостной сюжет, процедура на производственной линии — сам по себе объем кадров становится главным ограничением. Сверточные модели вынуждены либо агрессивно ужимать время, либо склеивать решения «кусочками», рискуя потерять дальние связи. Внимание, даже в упрощенном виде, позволяет держать длинную «нить повествования»: модель может напрямую связывать моменты «до» и «после», даже если между ними десятки секунд, и этим объясняется успех современных видеомоделей на задачах вроде «раскадровки» историй и сложных взаимодействий объектов. В то же время, когда нужно много и быстро — например, отслеживание людей в зале безопасности с высокой частотой кадров — аккуратно собранные 3D-свертки и «двухтемповые» схемы дают ту самую надежность и предсказуемую задержку.

Отдельно стоит сказать о данных. Видео разметка особенно дорогая: размечать границы событий по кадрам — дело не быстрое и каждый оператор размечает чуть по-разному. Поэтому так ценится самообучение: модель смотрит на километры немаркированного видео и учится предсказывать пропуски, сопоставлять соседние кадры, находить устойчивые признаки движения и формы. Такая «закалка» потом дает эффект на реальных задачах. Второй источник — синтетика. В спорте можно сгенерировать редкие комбинации, в производстве — смоделировать сбои, которые вживую трудно и небезопасно ловить. Главная цель — не фотореализм, а разнообразие: менять фон, освещение, скорость, ракурсы, чтобы модель перестала привязываться к случайным примерам и научилась видеть именно действие.

3D-восприятие: из плоского кадра к миру с глубиной

До сих пор мы рассуждали о том, «что» и «где» в пределах плоскости кадра. В трех измерениях вопросы становятся богаче: на каком расстоянии лежит поверхность, как устроена форма сцены, куда направлена и где находится сама камера, куда движется объект, как очертить его объем не только по контуру, но и в глубину. Формально нас интересуют карты глубины и диспаратности, геометрия трехмерных поверхностей, позиция камеры и объектов относительно выбранной системы координат, а еще — детекция уже в 3D: не просто рамки на картинке, а коробки в пространстве, у которых есть ширина, длина, высота и ориентация.

Две камеры смотрят на один и тот же мир с небольшой базой между ними (как наши глаза), и одна и та же точка сцены попадает на разные горизонтальные позиции в левой и правой картинке. Разность этих позиций называется диспаратностью, и она обратно пропорциональна глубине: чем больше сдвиг — тем ближе объект ($z \approx f \cdot B/d$, где f — «фокус», B — расстояние между камерами, d — диспаратность). Если же камера одна, мы полагаемся на другие закономерности: перспектива, размытие, параллакс при движении. Моноскопическая сеть учится угадывать глубину по миллионам примеров, а в видео связывает соседние кадры и минимизирует «фотометрическое несоответствие»: если мы правильно восстановили глубину и движение камеры, то сможем «перенести» один кадр в другой и получить очень похожую картинку. Это позволяет обучаться даже на данных без «золотых» карт глубины, но оставляет главную неоднозначность — абсолютный масштаб. Два одинаковых кадра могут соответствовать маленькой сцене, снятой крупным планом, и большой — с дальней точки; без внешней «линейки» (лидара, стерео, известных размеров объектов) модель в принципе не различит эти случаи.

Поэтому, единственный ключевой параметр, который я запрашиваю у пользователя моего бильярдного приложения при загрузке очередного видео, это физический размер бильярдного стола. Удобство заключается в том, что эти размеры типизированы. Их всего четыре и обычно они хорошо известны пользователю, без необходимости иметь измерительную рулетку под рукой.

Разметка в 3D дорожке и разнообразнее. Иногда есть «правильные» карты глубины — их дают специализированные сенсоры (лидар, времяпролетные камеры, структурированный свет). Чаще есть только частичная истина: разреженное облако точек от лидара, которое покрывает далеко не все пиксели, или 3D-коробки у автомобилей из набора для автономного вождения, где аннотации писались по совокупности

датчиков. В задачах одновременной локализации и построения карты (SLAM) у нас вообще нет «правильной» карты для каждого шага — мы проверяем себя постфактум, сравнивая восстановленную траекторию камеры с эталонной или же оценивая, насколько хорошо «сшиваются» разные проходы по одному и тому же маршруту.

Зачем все это? В роботах и дронах глубина — имеет ключевое значение: пол, ступенька, стенка стеллажа и пустой проем — разные классы не только по смыслу, но и по расстоянию. В автономном вождении 3D-детекция позволяет различать автомобиль, мотоцикл и пешехода не по силуэту на картинке, а по телу в пространстве: мы знаем габариты, ориентацию, скорость, можем безопасно планировать маневр. В дополненной или виртуальной реальности нам нужно «положить» виртуальный объект на реальный стол и удержать его там при движении камеры — без правдоподобной глубины и позиции камеры мы всегда будем получать «плавающий» фантом. В инженерных измерениях и реконструкции мы переводим мир в облака точек, сетки и воксели, строим цифровые двойники, считаем объемы и допуски.

Качество глубины обычно измеряют просто: сравнивают предсказанную карту с эталоном по среднему модулю ошибки или по среднеквадратичной. Но есть тонкости. Если масштаб не определен (моновидео без якоря), оценивают качества «с точностью до масштаба»: сперва подбирают коэффициент, выравнивающий общую «высоту» карт, и только потом считают ошибки. Для оценки качества 3D-детекции, результаты сводят в знакомые интегральные числа наподобие «средней точности» по наборам вроде KITTI или nuScenes, но совпадать должны уже объемы: коробка модели и истинная коробка имеют достаточную долю пересечения в пространстве, а еще учитывается правильность ориентации и классификации. В SLAM две базовых меры смотрят на траекторию: насколько далеко, в среднем, ушла восстановленная позиция от истинной (ATE — итоговая погрешность по траектории) и насколько стабилен относительный сдвиг между соседними шагами (RPE — локальная согласованность движения). Если ATE небольшой, но RPE большой — вы «блуждаете» на каждом шаге, а если наоборот — локально все гладко, но медленно «уползаете» из-за накопления ошибок.

Модельных семей несколько, и они отражают физические возможности имеющихся сенсоров. Для глубины из одной камеры проще всего начать с обучения по примерам: сеть принимает кадр и возвращает карту глубины, но далеко с этим не уедешь без больших датасетов с истиной. Более практичны схемы, где учителем служит сама геометрия камер: минимизируется расхождение между реальным и «пересчитанным» кадром, а параллельно оценивается движение камеры (его стабилизируют с помощью инерциального датчика). Для стереовхода хватает

двух кадров, и задача сводится к поиску сдвига вдоль линий параллакса; в обучении используются как готовые наборы с диспаратностью, так и синтетика, где истина известна идеально.

3D-детекторы учатся на облаках точек (с лидара) и на их связке с камерами: камера дает густую текстуру и помогает различать классы, лидар — метрическую опору и точную дальность, особенно ночью и в плохую погоду. Наконец, есть семейство «неявных» представлений сцены: вместо явной сетки или облака точек мы храним функцию, которая по координате и направлению взгляда возвращает «как много материи и какого цвета» там находится. Такая сцена рождается из фотографий с разных ракурсов через дифференцируемый рендеринг и потом позволяет синтезировать новые виды почти как в реальности. Цена — вычисления и требовательность к калибровке.

У 3D-восприятия свой набор проблем. Первая — масштаб в монокуляре. Без внешних подсказок модель может выучить убедительную относительную глубину («это ближе, это дальше»), но запутаться в абсолютных метрах. Простой способ притянуть масштаб — IMU, стерео или лидар; даже редкие «якоря» в виде известных размеров (полосы разметки, стандартный дорожный знак) помогают вернуть шкалу масштаба.

Вторая — отражающие свойства материалов. Зеркальные и блестящие поверхности «обманывают» и камеру, и активные датчики: лидарный луч уходит в блик, времяпролетная камера страдает от многолучевости, стекло «подставляет» отражения дальнего мира вместо ближнего. Здесь выручают многоракурсность и слияние модальностей: если камера видит блик, лидар может вернуть скудные, но честные точки; если лидар «промахнулся» на стекле, камера по текстурам «догадается» о плоскости. Полезны и прозаические приемы — поляризационные фильтры на оптике, сценарные ограничения (не доверять глубине на глянцевых полах, если угол падения мал).

Третья — синхронизация. Камера, лидар и инерциальный датчик «живут» в разных временных доменах; несогласованность на десятки миллисекунд уже дает смаз по движущимся объектам и кривые облака точек, особенно при поворотах. Без точной временной метки и аккуратной калибровки внешних параметров (где и как повернуты датчики относительно друг друга) слияние модальностей будет давать неверные картинки.

Четвертая — «симулятор против реальности». Учесть глубину и 3D в симуляции удобно: это бесплатно и идеально верифицируемо, но реальные шумы, смаз, туман, грязь на линзах и хаотическая текстура мира быстро разрушат лабораторные настройки.

Поэтому большое внимание следует уделить калибровке источников: внутренние параметры камер, положения сенсоров на платформе, временная привязка — все это надо измерять, проверять и отслеживать в процессе эксплуатации; небольшие дрейфы креплений на роботе после смены бампера уже заметны в картах глубины. Слияние модальностей работает, когда каждая модальность откалибрована: камера даёт опорную структуру окружающего мира, лидар — опорные точки, IMU — гладкую скорость и направление, и в общей задаче фильтр «доверяет» тому, кто в данный момент информативнее. В подходе который опирается только на видео, имеет смысл «якорить» масштаб по наблюдаемым эталонам и доучивать под конкретный парк камер: там своя оптика, скорость съёмки и свое поведение автоэкспозиции. А если приходится начинать в симуляции, то разумно «разнообразить» мир: рандомизировать текстуры, освещение, добавлять шумы сенсоров и обязательно доводить модель на небольшом реальном наборе — разница между «почти фотореалистичным» и «достаточно похожим» для нейросетей огромна.

Например — 3D реконструкция квартиры для визуализации ремонта в дополненной реальности. Хочется быстро «снять» помещение телефоном, расставить мебель и пройти по реальному пространству с наложенными моделями. Здесь оценка глубины камерой недостаточна стабильна для миллиметровой точности, но достаточна для сантиметровой. Помогает аккуратная траектория камеры: медленное, равномерное движение, повороты вокруг точек интереса, замыкание петель (возврат к уже виденным местам) — алгоритм узнает «старые» виды и это снижает накопленные ошибки и затягивает 3D сетку комнаты к реалистичной геометрии. Там, где необходима большая точность — например, для вписывания кухонного гарнитура, — мы добавляем линейку в кадр или хотя бы «вешаем» известный размер на элемент интерьера. И не забываем про «смысловую» часть: стену можно считать плоскостью, пол — тоже, двери — прямоугольниками, и такие допущения заметно улучшают результат даже при шумной глубине.

В 3D-восприятии геометрия неразрывно связана со статистикой. С одной стороны, у нас есть строгие соотношения проекций и движений, калибровки и триангуляции; с другой — шумные датчики, блестящие поверхности, динамичный мир. Лучшие системы не выбирают «или-или», они совместно оценивают все источники, учатся на реальных данных с недостающей истиной и опираются на физику там, где она сильнее. Выбор датчиков, представлений и метрик диктуется приложением, но принцип везде один: максимум правды от каждого источника, минимум неустойчивых узлов и бережное обращение с неопределенностью.

OCR

С документами у зрения другая задача: мы хотим не распознать кошку или дорогу, а «прочитать» изображение. Это чтение состоит из трех шагов. Сначала — понять, где на картинке вообще есть текст: выделить строки, слова, отдельные блоки вроде адреса отправителя или суммы к оплате. Затем — превратить эти участки в последовательности символов: уже не «текст тут», а конкретно «УЛ. ЛЕНИНА, 12». И, наконец, собрать смысловую структуру: кто из этих фрагментов — заголовок, что — таблица с товарами, где валюты и налоги, где подписи и печати, какова «логика» чтения страницы. По сути, мы идем той же дорогой, что и человек: взгляд цепляется за куски текста, мозг складывает буквы в слова, а затем узнает знакомые формы документа — «это чек, а вот строка с итогом».

Разметка отражает эти шаги. Аннотатор обводит текстовые регионы прямоугольниками или многоугольниками, чтобы учесть наклон и кривизну строк; к каждому региону приписывает точную транскрипцию — последовательность символов в том виде, как их должен выдать алгоритм (с пробелами, знаками, правильной раскладкой). Там, где важна «анатомия» документа, размечают роли: «платательщик», «ИНН», «сумма прописью», «таблица товаров» с границами ячеек и заголовками столбцов. Это трудоемко: аккуратная разметка одной накладной с таблицей может занять больше времени, чем десяток фотографий для детекции объектов. Но именно здесь закладывается устойчивость: если в эталонах перепутан порядок чтения, неправильно обрезаны края строк или роли полей определены невнятно, модель потом переносит эту небрежность на тысячи документов.

Сценариев применения — множество. В финансах и логистике речь про поток: чеков и инвойсов много, и важно автоматически извлекать суммы, НДС, номера счетов, коды товаров и контрагентов, чтобы системы учета не зависели от ручного ввода. В госуслугах и банке клиент фотографирует паспорт; от того, насколько надежно распознается зона с машиночитаемой строкой и кириллическими фамилиями, зависит скорость обслуживания. В архиве нужно оцифровать десятки тысяч страниц, сделать их полнотекстовыми для поиска, не теряя при этом структуру: заголовки, колонтитулы, подписи к иллюстрациям. В корпорации — найти все договоры с условием об автоматическом продлении: без понимания текста и структуры этого не сделать.

Качество здесь измеряют по-разному, и каждое число отвечает на свой вопрос. Для детекции текстовых блоков важна точность в духе «сколько истинных блоков мы нашли и сколько лишних нарисовали», а совпадение оценивается по доле пересечения границ (как в детекции объектов, но с поправкой на вытянутые, наклонные и кривые строки).

Для распознавания смотрят на долю ошибок в символах и словах: это прямой ответ на вопрос «сколько, в среднем, букв испортили на сто символов». В задачах извлечения ключевых полей уместнее думать в терминах «сколько нужных значений нашли и правильно ли они записаны», ведь «ИНН: 7707083893» либо совпадает с истиной, либо нет; для таблиц дополнительно оценивают правильность структуры — не напутали ли строки и столбцы, верно ли связали числа с заголовками. И всегда добавляют измерение устойчивости: ведь документ может быть снят под углом, в сумерках, с бликом, с печатью поверх текста и с полустертой подписью.

Как это работает изнутри, на уровне идей. Модули локализации «смотрят» на картинку и учатся находить длинные узкие регионы, где есть регулярные штрихи и контраст, характерный для букв и цифр (детекторы текста вроде EAST/DB). Они умеют учитывать наклон и кривизну строк, а также отделять текст от похожих на него узоров. На следующем шаге изображение строки превращается в последовательность символов. Исторически это делали, разрезая строку на отдельные буквы, но это хрупко: в реальности буквы сливаются, прилегают, касаются штампами. Современный подход идет без разрезания: он смотрит на строку как на «ленту» и по мере продвижения считывает буквы, согласуя видимое с последовательностью (в этой роли часто выступает выравнивание «из картинки в символы» без точной привязки по кадрам; альтернативный путь — «внимание», которое подсвечивает текущий фрагмент строки под предсказываемым символом). На уровне документа подключается «чувство макета»: модель учится понимать, где логические блоки, как они выстроены в колонках, что похоже на таблицу даже без явных линий, а что на подпись к печати. Мультимодальные модели добавляют к этому язык: слово «ИТОГО» с большой вероятностью соседствует с суммой и знаком валюты, а «SHIP TO» в инвойсе влечет адресный блок.

Проблемы тоже ожидаемы. Первая — грязь реального мира: смятые чеки, сканы с перекосом, фотографии страниц под углом, тени от пальцев, пыль на стекле МФУ. Все это «ломает» регулярность штрихов и цвет буквы. Самая дешевая защита — доброкачественная предобработка: выровнять страницу по перспективе, приглушить шум и пульсации фона (denoise), исправить неравномерное освещение, удалить цвет фона, если он мешает контрасту. Вторая — печати и штампы поверх текста. Человек легко «фильтрует» их, а машине нужна помощь: или обучающие примеры с печатями поверх разных шрифтов, или методы, которые учатся отделять «плотные круги/прямоугольники» печатей от тонких штрихов букв, а иногда просто вторая попытка распознаваться в негативе — белым по темному. Третья — рукопись. Печатные шрифты отличаются умеренной вариативностью, а почерк — безбрежный. Тут

спасают два фактора: объем данных и язык. Если задача — распознать фамилию на бланке с небольшой алфавитной «стороной» (латиница/кириллица цифры), можно специально учить на рукописях и ограничить предположения словарем и шаблонами («Фамилия Имя Отчество»). Общие рукописные тексты — тяжелая область, и хорошая практика — честно показывать неопределенность, предлагать несколько вариантов и выносить сомнительные фрагменты на проверку человеком. Четвертая — многоязычие и смешанные алфавиты. В одном документе легко уживаются кириллица, латиница, цифры, спецсимволы. Нужен либо общий «алфавит», либо шаг определения сценария: сначала понять, каким письмом написан фрагмент, потом распознавать в соответствующем режиме. Языковые модели и словари здесь не «магия», а страховка: они подсказывают, что «СУММА» — слово из русского делового лексикона, а «€» редко соседствует с «руб.».

Структура — отдельная тема. Таблица может быть с рамками, без рамок, со слипшимися ячейками, с заголовками над несколькими столбцами. «Видеть» таблицу — значит не только найти числа и слова, но и восстановить логические ряды: какая цена относится к какой позиции, где количество и единица измерения. Обучать этому можно по примерам с разметкой ячеек и связей; можно помогать классическими приемами — анализом линий, сеток, межстрочных промежутков. На практике часто выигрывает гибрид: нейросеть находит кандидатов в строки и столбцы, дальше вступают простые правила, которые проверяют внутреннюю согласованность («в столбце цен должны быть числа с двумя десятичными знаками»). Это вдвойне важно, если документ «гуляет» от поставщика к поставщику: один ставит «ИТОГО» вверху, другой — внизу, у третьего таблица на двух страницах.

Без постобработки и валидации не обходится ни одна промышленная система. Если распознан «ИНН», но контрольная сумма не сходится, есть шанс, что перепутались похожие символы и нужно попробовать еще раз или попросить человека заглянуть. Для дат, сумм, кодов удобно использовать регулярные шаблоны и проверять согласованность: валюта в шапке и валюта в строке итога обязаны совпадать, налог не может превышать сумму. Для адресов и организаций помогают справочники и внешние базы: строка «ул. Ленина, 12» превращается в нормализованный адрес, а «ООО Ромашка» — в запись с ИНН и ОГРН. Главное — не переусердствовать: жесткие правила легко ломаются на реальных разнообразных документах, и система должна уметь аккуратно снижать уверенность там, где она не уверена, а не придумывать «правду».

Стоит сказать и про приватность. Документы несут персональные и финансовые данные, и архитектура решения — не абстрактный вопрос. Где будут храниться изображения и результаты? Кто и как имеет доступ к сомнительным полям для верификации? Локальная обработка на устройстве или в пределах периметра компании часто важнее, чем на несколько процентов выше точность в облаке. Если без человека не обойтись (а в начале пути без него не обойтись), то интерфейс для проверки должен быть продуманным: подсветка областей с низкой уверенностью, быстрые горячие клавиши, мгновенная навигация по документам. Это экономит часы и снижает вероятность механических ошибок.

Наконец, две истории, которые показывают разницу между «читать» и «понимать документ». В ритейле внедряли автоматический учет расходов по чекам сотрудников. Сначала казалось, что достаточно распознать текст и вытащить сумму. В пилоте стало ясно: у одного магазина «итог» пишется как «ИТОГО», у другого — «ВСЕГО», у третьего — «СУММА К ОПЛАТЕ», а еще есть скидки, возвраты и бонусы. Модель научили не просто искать число, а узнавать «паттерн итога» в контексте: слово-маркер рядом, сумма визуально выровнена по правому краю, формат числа соответствует денежному формату. Добавили проверку согласованности с суммой по строкам таблицы — и число ложных «итогов» упало в несколько раз. Во втором проекте — в логистике — из международных накладных нужно было извлекать адреса и веса. Документы приходили с факса, с кривыми страницами и печатями. И здесь выстрелила простая дисциплина: приложение для съемки на складе подсказывало, как держать телефон, автоматически выравнивало перспективу, предлагало доснять край. Даже без изменения моделей доля нечитаемых зон уменьшилась почти вдвое, а «грубая» калибровка яркости перед отправкой на распознавание спасла светлые бледные строки от потери.

ЧАСТЬ 6

Взгляд в будущее

Я долго сопротивлялся самой идее этой главы. Дело в том, что мы уже подробно поговорили про то, что можно хорошо измерить, ускорить, улучшить, воспроизвести. Про датасеты, нейросетевые архитектуры, машинное обучение, калибровку камер, ошибки разметки, странные провалы модели в «простых» сценах, неожиданную хрупкость, которая хоть и близка к порогу устойчивости, но дает не нулевую вероятность ошибки в самый неподходящий и не обоснованный момент. Сталкиваясь с неустойчивыми результатами и ошибками, мы вынужденно приходим к теоретическому обобщению — выстраиваем против них конструкции из новых понятий гипотез и экспериментов.

И вот поэтому я предлагаю читателю шагнуть в сторону — не от практики, а поверх нее. Эта глава о том, что любая инженерная дисциплина, если она действительно вырастет, рано или поздно находит собственную внутреннюю философию как способ держать всю задачу целиком, не теряясь в частностях. В машинном зрении особенно ясно видно, почему без такого уровня нельзя: мы слишком легко начинаем путать улучшение отдельных модулей с пониманием системы. Здесь очень к месту вспоминается наследие Дэвида Марра — его идея трех уровней объяснения. Можно бесконечно совершенствовать вычислительные механизмы и даже описывать алгоритмы все точнее, но без ответа на вопрос “что именно мы вычисляем и зачем” вся эта точность остается локальной. Теория нужна как мост между целями, алгоритмами и практикой: чтобы различать, где мы действительно приближаемся к решению, а где просто научились подгонять алгоритмы под желаемый результат.

Мы привыкли рассказывать историю машинного зрения как историю инструментов. Сначала — анализ геометрии фрагментов, затем — детекторы признаков, затем — статистические модели, затем — глубокие нейросети. Потом трансформеры, самоконтролируемое обучение, диффузии, мультимодальность. Все это можно описать так, как я и описывал до сих пор: как смену поколений методов. Но если семь лет подряд жить внутри этих систем — читать обзорные статьи, своими руками строить продукты, чинить сбои, ловить баги в данных, спорить с реальным миром, который никогда не бывает «как в датасете» — то рано или поздно начинаешь чувствовать, что кроме технологической линии развития существует еще одна, менее заметная: понятийная линия того, что мы считаем объяснением наших гипотез.

В начале пути мне казалось, что объяснение — это когда я могу разложить проблему на блоки. Камера дает картинку. Дальше — детектор. Дальше — трекер. Дальше — фильтр. Дальше — логика. Если что-то не работает, значит где-то блок слабый: надо подкрутить параметры, собрать больше данных, сменить архитектуру. Это классический инженерный подход.

Но по мере того как модели становились мощнее, а алгоритмы — короче, я все чаще ловил странное ощущение: мы строим систему, которая работает, но объясняем ее проще, как будто это случайность, удачное попадание в гиперпараметры и масштаб. И наоборот: когда система ломалась, ломалась она часто не там, где мы ожидали. Не на «сложном», а на простом. Не на редком, а на естественном. На мой взгляд, это один из явных симптомов того, что в нашем языке описания есть дырка.

До какого-то момента эту дырку можно зашивать практикой: добавить данных, усилить аугментацию, поправить разметку, ввести неуверенность, и т.п. Я делал это десятки раз. И каждый раз система становилась лучше. Но одновременно становилась еще более похожей на живое существо: у нее появлялись «привычки», «предпочтения», «слепые зоны», «интуиция». Я не романтизирую нейросети — это просто мое ощущение того, как ведет себя высокоразмерное представление, обученное на мире, который сам по себе неоднозначен.

И вот в какой-то момент я поймал себя на мысли: возможно, мы слишком долго считали, что машинное зрение — это про распознавание объектов. Потому что объект — удобное слово для человека, но не обязательно фундаментальная единица для модели. Мы строим интерфейс к реальности через «классы», «боксы» и «маски», но внутри сети — чистая математика. Там нет стула как слова. Там есть устойчивые направления в пространстве представлений. Там есть математические узоры, которые повторяются в разных сценах, но не сводятся к словам. Там есть такие «смыслы», которые проявляются в динамике, в ошибках, в том, как модель переносит знание между доменами. Я долго отмахивался от этого как от метафоры. А потом понял: это не метафора. Это подсказка.

Парадокс машинного зрения последних лет в том, что мы одновременно стали ближе к «универсальному восприятию» и дальше от человеческого объяснения. Мы научились строить модели, которые видят почти все, но объясняем их через термины, унаследованные от классической компьютерной графики и когнитивной психологии: «объект», «форма», «контур», «текстура», «сцена», «понимание». Эти слова полезны, но они слишком грубые. Это как пытаться описывать музыку только по

названиям инструментов. Да, можно, но вы теряете главное: внутренние отношения, напряжения, резонансы, переходы.

Именно поэтому я захотел сделать эту последнюю главу. Это попытка вернуть себе — и, если получится, читателю — чувство целостности. Мне кажется, что лучший способ взглянуть в будущее — это довести инженерную интуицию до абстракций, которые позволяют лучше понимать модели. Я пишу это как человек, который семь лет жил в цикле «наблюдение — обучение — разочарование — исправление — снова наблюдение». И за эти годы во мне накопилось несколько мотивов, которые хотелось бы описать подробнее.

Первый — профессиональная усталость от поверхностных объяснений. Это не упрек кому-то конкретному. Просто мы как специалисты отрасли привыкли заменять понимание масштабом: если модель большая и датасет огромный, она будет работать. Работает — и отлично. Но когда ты отвечаешь за систему в реальном мире, одной фразы «оно выучится» недостаточно. Тебе нужно заранее понимать, где она сломается, как она перенесет новый домен, какие ошибки будут «опасными», а какие — косметическими. Мне хотелось языка, который объясняет не только среднее качество, но и характер ошибок.

Второй мотив — желание собрать в одну теорию то, что обычно разнесено по разным дисциплинам: восприятие, представление, действие, симуляцию, генерацию. Мы привыкли думать, что генерация изображений — это другое направление, отдельная ветвь. Но последние годы показывают: эти части срастаются. Модель, которая видит, начинает моделировать мир; модель, которая моделирует, начинает планировать; модель, которая планирует, начинает генерировать; а генерация все чаще уходит из пикселей в механику — в роботов, производство, автономные системы. Мне хотелось описания, в котором этот цикл естественен.

Третий мотив — почти личное переживание того, как меняется ощущение реальности, когда ты долго смотришь на мир глазами модели. Это странная штука, и я не уверен, что ее поймет каждый. Но если вы годы подряд смотрите на сцену через ошибки детектора, через эмбединги, через латентные пространства, через уверенности и неопределенности, вы начинаете замечать: мир для модели не состоит из «вещей» в человеческом смысле. Он состоит из устойчивых различий. Из тех самых различий, которые выживают при шуме, которые предсказывают дальнейшее, которые можно использовать, чтобы действовать. И в какой-то момент ты понимаешь, что это не компьютерная точка зрения. Это просто более строгая — математическая интерпретация окружающей

действительности. Она иногда кажется бесчеловечной, но зато она более объективная: она показывает суть вещей, которая существует независимо от субъективной точки зрения.

Поэтому дальше я сделаю в книге то, что обычно не делают в технических текстах: я предложу сменить точку опоры. Не отказаться от всего предыдущего, а переосмыслить это как частные случаи более общей картины. Я буду двигаться от того, что мы уже знаем и используем — латентные пространства, инвариантность, эквивариантность, самообучение, мировые модели, генеративные подходы — к более абстрактному описанию, которое позволяет говорить о восприятии, понимании и генерации как о разных аспектах одного и того же процесса.

Я понимаю, что на этом месте у читателя может возникнуть сопротивление. “Зачем вообще нужна новая абстракция, если модели и так работают?” Я бы ответил: потому что мы подошли к границе, где просто «еще больше данных и параметров» перестает быть достаточным объяснением. Работать будет, но мы не будем понимать цену, пределы, риски и направления. Мне хочется, чтобы у нас появился язык, который не отстает от возможностей технологий.

Гипотеза внутреннего «пространства качеств» в нейросетях

Под «пространством качеств» понимается многомерное пространство устойчивых факторов восприятия модели. Эти факторы – не произвольный «латентный код», а осмысленные оси, соответствующие стабильным свойствам окружающего мира. Важное требование: такие внутренние координаты должны сохранять смысл при изменении условий наблюдения, согласовываться друг с другом и помогать модели точнее предсказывать поведение среды во времени. Речь идет не о простом «наборе признаков», а о внутренней системе координат модели, где каждая ось – некоторый устойчивый качественный признак сцены: например, освещенность, материал поверхности, глубина (расстояние до объекта), подвижность, проходимость и т.п. В таком представлении изменение внешних условий (угла обзора, освещения, шума) сдвигает точку в этом пространстве по соответствующим осям, но не разрушает ее базовую структуру. По сути, моя гипотеза утверждает, что успешные глубокие нейросети не просто запоминают шаблоны объектов, а вычленивают базовые свойства среды, обеспечивая разделение факторов. Идеи, близкие к этому, встречаются в работах по дизентанглменту (disentanglement) – например, в модели β -VAE усиливается разделение независимых факторов вариации в скрытом пространстве, так что отдельные компоненты латентного вектора начинают отвечать за понятные изменения изображения (например, поворот объекта или масштаб).

Архитектура β -VAE (бета-вариационный автоэнкодер) была предложена как модификация классического VAE с целью достичь *disentanglement* — разделения скрытых факторов в представлении. В основе β -VAE лежит идея усилить регуляризацию латентного пространства, увеличив вес KL-дивергенции (различие между аппроксимируемым постериорным распределением и нормальным априорным) в функции потерь. Этот вес обозначается буквой β : если в стандартном VAE $\beta = 1$, то в β -VAE он становится больше 1. Это принуждает модель сильнее приближать распределение скрытых переменных к независимому гауссовскому, тем самым снижая «избыточность» между ними. В результате, если структура данных и архитектура позволяют, каждый латентный фактор начинает отвечать за независимый аспект наблюдения: положение, масштаб, поворот, форма, и т.п. В визуальных экспериментах с синтетическими наборами (например, dSprites) β -VAE показывает, как при варьировании одного скрытого параметра изменяется, скажем, только форма объекта, а остальные остаются фиксированы. Однако эта архитектура — компромисс: слишком высокий β ухудшает реконструкцию, и *disentanglement* возможен только при наличии некоторой индуктивной структуры в данных. Тем не менее, β -VAE стал отправной точкой для

целого класса моделей, в которых цель — не просто кодировать данные, а сделать структуру представления понятной и управляемой.

Однако “качественное пространство” — более глубинное понятие: оно акцентирует именно смысловые качества восприятия, которые модель вынуждена выделять для устойчивого прогнозирования мира.

Почему такой подход важен? Дело в том, что некоторые представления, которые вырабатывает модель, оказываются чрезвычайно универсальными — они переносятся между разными задачами и доменами, обеспечивая обобщающую способность. Другие же, напротив, рассыпаются при малейшем отклонении условий. Гипотеза о внутреннем пространстве качеств призвана объяснить, **почему** одни признаки оказываются надежными и переносимыми, а другие — нет. Если внутренняя координата модели соответствует реальному устойчивому смыслу сцены, она будет актуальна в разных ситуациях (например, понятие “освещенность” важно и днем, и ночью, и в помещении, и на улице), и модель сможет использовать ее повторно. Если же признак привязан к частному шаблону (например, конкретная текстура, видимая только при одном освещении), то небольшое изменение условий сделает его бесполезным. Таким образом, внутреннее пространство качеств служит своего рода «картой местности», дополняющей карту инструментария (архитектурных приемов), о которой речь шла ранее. Оно объясняет, почему некоторые архитектурные механизмы оказались необходимыми: они способствуют формированию правильных координат и связей в этом пространстве. Ниже мы подробнее раскроем эту гипотезу, опираясь на современные представления о работе нейросетей и сопоставляя ее с известными теоретическими концепциями в области компьютерного зрения и обучения представлений.

Локальные качественные градиенты: свертки как измерители изменений

Начнем с того, что хорошо известно и привычно: классическое машинное зрение исторически было «объектно-ориентированным». Изображение рассматривалось как набор объектов и границ между ними. В традиционных алгоритмах искали края, углы, текстуры; строились дескрипторы (SIFT, HOG и др.), находились соответствия между изображениями, оценивались геометрические преобразования, восстанавливалась поза объекта. Появление сверточных нейросетей сначала казалось прямым продолжением этой логики: первые сверточные слои выделяют простые градиенты, последующие – собирают их в более сложные формы и шаблоны объектов. Однако более внимательное изучение внутренних активаций сетей показало иную картину. Если наблюдать не сами пиксельные картинки, а динамику внутренних признаков сети при изменении условий (скажем, при появлении бликов, теней или при переносе на новые домены), видно, что сеть не столько находит готовые объекты, сколько собирает мир из устойчивых различий и переходов. Первичными элементами оказываются не именованные объекты, а качественные признаки и то, как они изменяются при допустимых трансформациях наблюдения.

Рассмотрим работу сверточного фильтра с этой точки зрения. Свертка (convolution) в предлагаемой гипотезе – это не просто поиск заданного шаблона, а локальное измерение изменения какого-либо качества. Фильтр (ядро свертки) отвечает на вопрос: *«насколько сильно здесь изменяется определенное свойство?»*. Например, один фильтр может реагировать на вертикальный градиент яркости (обнаруживая вертикальный край), другой – на переход от одного цвета к другому, третий – на локальную текстурную шероховатость. Многие исследования отмечали, что первые слои CNN действительно учатся похоже на набор простых фильтров Габора и детекторов контрастных цветовых переходов – по сути, они выявляют ориентированные границы и разрывы в изображении. Эти градиенты качеств формируют на выходе первого сверточного слоя множество карт признаков, которые можно интерпретировать как поле интенсивностей разных качеств по всей сцене. Каждая карта активаций показывает, где в изображении сильно проявлено соответствующее качество (будь то контур определенной ориентации, текстурный паттерн или иной локальный признак).

Далее вступает в игру агрегация – свертки более высоких уровней и операции pooling усредняют или обобщают локальные проявления качеств, делая представление все более инвариантным к несущественным микродеталям. Логика иерархии здесь следующая: сначала фиксируются самые грубые перепады (первичные грани и текстуры), затем

нейросеть учится не реагировать на мелкие сдвиги, повороты, масштаб – т.е. формируются признаки, устойчивые к таким трансформациям, – и, наконец, на более глубоких уровнях вычленяются все более абстрактные устойчивые структуры, которые переживают шум, изменение домена и прочие вариации. В классических работах отмечалось, что глубокие сверточные сети действительно выстраивают подобную пирамиду признаков: от низкоуровневых границ и цветов – к углам, формам, текстурам – и далее к сложным сочетаниям, характеризующим целые объекты или их части. Можно сказать, что глубина сети – это число шагов “очистки” координатного пространства от лишних вариаций: каждый следующий слой абстрагируется от все большего количества деталей. Благодаря этому меняется само качество признаков – например, на уровне пикселей тень от облака радикально меняет значение каждого пиксела, но на высокоуровневой активации, соответствующей “объекту”, та же тень почти не влияет на вывод. Тот факт, что глубокая сеть может приобрести такую устойчивость, имеет и строгие теоретические обоснования: в работе «Invariance and Stability of Deep Convolutional Representations» показано, что многослойные сверточные представления способны быть почти инвариантными к определенным трансформациям входа, оставаясь при этом информативными. Иными словами, сеть выстраивает пространство признаков, в котором малые изменения изображения (повороты, небольшие деформации) отражаются слабо (Lipschitz-стабильность), а значимые различия сохраняются.

Таким образом, уже на уровне локальных операций вырисовывается другая парадигма: сеть оперирует не прямыми образами объектов, а полями «качеств». Объект как бы «собирается» из устойчивых локальных сигналов. Это согласуется с тем, что обнаружили Geirhos и коллеги (2018) при изучении поведения CNN, обученных на ImageNet: сети оказались сильно смещены в сторону локальных текстур, а не целостных форм объекта. В то время как человек узнает предмет в первую очередь по контурам и общей форме, сверточная сеть склонна хвататься за мелкие частные детали – например, характер шерсти или пятен на шкуре – которые отличают один класс от другого в обучающих данных. Это означает, что для нейросети важнее стабильные локальные различия (текстуры), чем глобальная геометрия формы – по крайней мере при стандартном обучении на классификацию. Однако такое поведение не является неизбежным: стоит изменить данные и цель обучения, и та же архитектура начнет выделять более глобальные признаки (форму). В том же эксперименте было показано, что обучение ResNet-50 на специально стилизованном датасете (Stylized-ImageNet), где убрана корреляция между текстурой и классом, приводит к значительно более выраженному учету формы – и это дает выигрыш в обобщающей способности и устой-

чивости модели. Это отличный пример того, как широкий разброс условий и принуждение к инвариантности заставляют сеть сориентировать свои внутренние оси на более фундаментальные качества изображения (такие как форма объектов), игнорируя частные «стили». В реальных задачах подобный эффект достигается и за счет простого масштабирования объема и разнообразия данных: большие и разнообразные корпуса обучающих изображений невольно учат модель не обращать внимания на нестабильные мелочи, вычлняя именно те факторы, которые повторяются повсюду. При сильной вариативности данных стратегия запоминания частных шаблонов просто не работает – сеть вынуждена обобщать. Именно поэтому увеличение широты и случайности обучающего набора, включая агрессивные аугментации и рандомизации домена, повышает устойчивость моделей к новым условиям. Фактически, методика рандомизации домена (широко применяемая, например, при обучении робототехники в симуляторе) специально генерирует изображения с *произвольными* вариациями цвета, текстур, освещения и т.п., чтобы модель научилась фокусироваться на инвариантных, существенных признаках среды.

Подведем промежуточный итог. На локальном уровне нейросеть строит множество «слоев качества»: каждый фильтр детектирует определенный тип различия, и совокупность таких детекторов образует базис для описания сцены в терминах интенсивности качеств. Более глубокие слои агрегируют эти интенсивности, формируя все более устойчивые признаки, мало зависящие от мелких вариаций. Такой взгляд позволяет по-новому интерпретировать привычные механизмы: например, свертка и полиинтеграция (pooling) – это не просто способы уменьшить размерности, а процедура перехода к новым координатам, очищенным от шума и несущественных деталей. Но локальных качественных признаков и их инвариантных комбинаций недостаточно, чтобы понять всю сцену – ведь объекты и ситуации определяются еще и взаимосвязью частей по смыслу. Для объединения разбросанных фрагментов восприятия в целостную картину нужен глобальный механизм согласования. Эту роль, согласно моей гипотезе, выполняет механизм внимания (attention).

Глобальная интерпретация: внимание и «синтаксис качеств»

Классические свертки и полиинтеграции имеют жесткую локальную привязку: нейрон в каждом следующем слое смотрит на фиксированное окно предыдущего слоя. Это ограничивает поле зрения модели: чтобы охватить контекст целиком, требуется много уровней или специальные структуры (как, например, пирамида признаков в FPN). Кроме того, локальные фильтры не умеют гибко устанавливать связи между дальними частями изображения – даже если эти части очевидно относятся к одному объекту или событию. Механизм внимания (self-attention), появившийся сначала в обработке текста, а затем успешно перенесенный в зрение (ViT, DETR и др.), решает эту проблему. Внимание динамически строит граф взаимосвязей между элементами представления в зависимости от их содержания. Проще говоря, каждый фрагмент изображения (например, патч или пиксельный признак) получает возможность *обратиться* ко всем остальным и «спросить», есть ли у них нечто общее или связующее. Это не топологически фиксированная (как свертка) связь, а смысловая: если два участка изображения содержат схожий признак или дополняющие друг друга качества, внимание может связать их напрямую, каким бы дальними они ни были. Благодаря этому Vision Transformer и подобные архитектуры с самого начала обладают глобальным контекстом – каждая часть изображения может учитывать информацию любой другой части.

Зачем это нужно? Чтобы собрать из локальных качественных «кубиков» глобально непротиворечивые структуры – по сути, ввести некий «*синтаксис качеств*», определяющий, какие сочетания признаков допустимы, а какие нет. В сценах реального мира множество признаков связаны закономерностями: если определенные края имеют одинаковое движение и материал, скорее всего, они принадлежат одному объекту; если участок дороги плавно переходит в обочину, у них будет согласованная геометрия и текстура и т.д. Классический пример – задача сегментации: один и тот же автомобиль на изображении может быть разорван тенью или бликом на нескольких отдельных фрагментах, хотя по смыслу это единый объект. Механизм внимания способен обнаруживать совместимость частей и объединять их: он увидит, что оба фрагмента имеют общее свойство «металлической поверхности», находятся на одном расстоянии (глубине) и движутся вместе, – следовательно, вероятно, это целое. Напротив, если соседние пиксельные области слабо разделены границей, но их качественные параметры резко различаются (скажем, одна – ближняя и статичная, а другая – дальняя и движущаяся), внимание *не позволит* «склеить» их ошибочно. Тем самым осуществля-

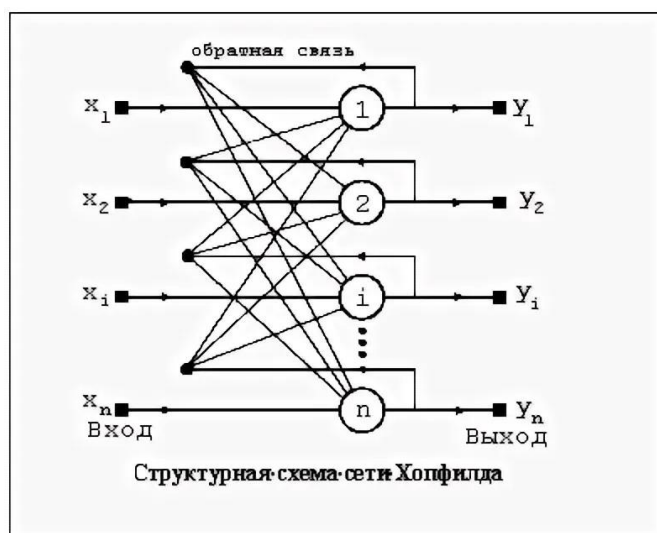
ется неявная группировка элементов по смысловым признакам – аналогично тому, как работает гештальт-принцип общей участи (common fate) или сходства в человеческом восприятии: части, движущиеся единообразно, мы видим как единый объект, а фрагменты схожей фактуры – как относящиеся к одной поверхности. Архитектурно это проявляется в том, что механизм внимания строит своего рода кластеризацию признаков: по итогам нескольких слоев внимания некоторые группы активаций сильно взаимодействуют внутри себя и слабо – с остальными, что указывает на их объединение в более крупные сущности (объекты или их части).

Стоит подчеркнуть, что внимание делает возможной гибкую многомасштабную проверку сцены. В современных сетях внимания (например, Vision Transformer с пирамидальными признаками, Swin Transformer и др.) есть механизмы, позволяющие учитывать соответствия на разных масштабных уровнях. Например, грубо очерченные крупные области согласовываются на высоком уровне пирамиды, а мелкие детали – на более низком, причем информация стекается и снизу вверх, и сверху вниз (через skip-connections или cross-attention между уровнями). По сути, модель может одновременно разгадывать синтаксис на нескольких уровнях абстракции: проверять, какие локальные градиенты согласуются в линии, какие линии образуют контур объекта, какие объекты совместимы между собой в данной сцене. Классические архитектуры FPN (Feature Pyramid Networks) в комбинации с блоками внимания можно рассматривать как инженерное приближение к такой системе: пирамиды признаков и перекрестные связи предоставляют каналы для обмена информацией между масштабами, а механизм внимания и схожие с ним не локальные слои явно улучшали точность детекции и сегментации за счет учета дальних связей (работа “Non-local Neural Networks”).

В итоге глобальный этап обработки – через внимание или иные виды промежуточного представления сцены – позволяет модели построить цельную интерпретацию наблюдаемого. Во внутреннем пространстве качеств это соответствует возникновению композитных координат, отражающих совместные конфигурации свойств. Если свертки задали “словарь” локальных качеств, а агрегация сделала их устойчивыми, то внимание задает правила их сочетаемости – неявно формируя представление о том, какие комбинации признаков соответствуют реальным объектам и ситуациям, а какие – нет (тот самый «синтаксис качеств»).

Резонанс качеств: устойчивые гипотезы и до- страивание недостающего

Интересным следствием описанных механизмов является способность модели додумывать скрытые части и подавлять помехи не путем перебора шаблонов из памяти, а через внутреннее согласование качеств. Когда сеть выучила устойчивые оси пространства признаков и научилась связывать фрагменты по совместимости, она фактически получила возможность восстанавливать недостающие данные на основе внутренней модели мира. Например, если объект частично окклюдирован, сеть уже видела подобные ситуации и может, опираясь на видимые качества (материал, тип контуров, характер движения), заполнить пробелы наиболее правдоподобным образом. Это проявляется как стабильность интерпретации: небольшие возмущения на входе не приводят к радикальному скачку во внутренних гипотезах – наоборот, активации «пружинят» обратно, сохраняя прежнюю идентичность объекта. Это явление называется резонансом качеств. Суть в том, что целая группа признаков начинает взаимно поддерживать друг друга, усиливая глобальную гипотезу. Ни один отдельный признак не главенствует жестко – важна именно связка координат, которая вместе образует устойчивый образ. Такая связка, однажды сложившись, энергетически выгодна для сети: ей соответствует минимум энергии ошибки или максимум правдоподобия, и системе проще слегка скорректировать несколько признаков, чем переключиться на совершенно другую интерпретацию при незначительном шуме. В терминах физических аналогий можно сказать, что сеть переходит в аттракторное состояние: как маятник, выведенный немного из положения равновесия, возвращается, так и внутренняя активация возвращается к консистентной конфигурации после малого возмущения.



Замечу, что идея ассоциативного восстановления паттернов не нова в нейрокомпьютинге. Классические сети Хопфилда (1982) демонстрировали, как распределенная память может хранить несколько стабильных состояний-образов, к которым сеть возвращается из произвольных близких начальных состояний. В них неполный или зашумленный образ автоматически дополняется до ближайшего хранимого – благодаря тому, что динамика сети спроектирована на минимизацию специальной энергетической функции. Такие устойчивые образы соответствуют локальным минимумам энергии (аттракторам), и эволюция сети всегда стремится к одному из них («Hopfield Networks in AI. Unlocking the Brain's Secrets: A Deep... | by Muhammad Abdullah Khan | Medium»). Другими словами, сеть «помнит» несколько шаблонов в виде устойчивых резонансных комбинаций активаций, и даже если мы частично затрем образ, система по оставшимся признакам восстановит его, притягиваясь к ближайшему аттрактору («Recurrent computations for visual pattern completion – PMC»). Современные глубокие нейросети не являются явно сетью Хопфилда, однако механизмы внимания и многослойной обратной связи придают им черты аттракторных систем. В литературе последних лет прослеживается возрождение интереса к таким феноменам: например, предложены новые формы слоев, эквивалентные непрерывной сети Хопфилда, которые позволяют интегрировать в архитектуру свойство ассоциативной памяти (Krotov & Hopfield, 2020). В контексте моей гипотезы «резонанс качеств» – это именно способность сети возвращаться к целостной гипотезе при частичном нарушении входных данных, причем гипотеза задается не меткой из словаря, а конфигурацией внутренних координат (освещенность, форма, положение, и т.д.). Такой подход сродни известному принципу «анализ через синтез» (analysis-by-synthesis) в компьютерном зрении: система понимает сцену, пытаясь

сама сгенерировать (синтезировать) ее объяснение и проверяя, согласуется ли оно с наблюдением. Если согласуется — значит, гипотеза верна и стабильна; если нет — гипотеза отбрасывается в пользу другой.

В бильярдном проекте я буквально проверял понимание сцены в соответствии с принципом «анализ-через-синтез». Для каждого эпизода удара я ставил физическую гипотезу: начальные позиции и скорости шаров брал из первых кадров трекинга, переводил пиксели в плоскость стола по калибровке углов и бортов, задавал коэффициент трения, упругость бортов и затухающие спины, после чего прогонял гипотезу на симуляторе на шаге, сопоставимом с частотой видео. Дальше сравнивал синтетическую траекторию с наблюдаемой: если по ходу эпизода они укладывались в узкий коридор по расстоянию и времени (ошибка порядка сантиметров и пары кадров), гипотеза считалась рабочей и фиксировалась; если же симуляция показывала невозможные вещи — шар ускорялся без причины, выходил из борта под неверным углом, «проезжал» через другой шар — я такую интерпретацию отбрасывал и возвращался к альтернативам: проверял неверный re-ID, переидентификацию шаров, пропуск столкновения, ошибку времени контакта. В случаях с недостающими данными — окклюзия, блик, низкий фреймрейт — синтетический эпизод помогал восстановить пропавший сегмент: по законам сохранения импульса и с учетом потерь на трение симуляция достраивала участок траектории так, чтобы вся сцена оставалась физически согласованной. По сути, система сама предлагала объяснение, а я принимал его только тогда, когда «симуляция удара» совпадала с тем, что видит трекер: согласуется — оставляем, не сходится — меняем гипотезу или помечаем эпизод как некорректный. Это дало не только фильтр качества данных, но и способ стабилизации: там, где видео подводит, физически правдоподобная симуляция удерживает интерпретацию в границах реальности. Если бы я мог встроить такую симуляцию прямо в нейросеть, точность распознавания эпизодов ударов на видео приблизилась бы к абсолютно человеческому значению. Для этого модель должна понимать качества окружающей действительности и законы природы на фундаментальном уровне.

В практических приложениях именно **резонансные связи качеств** определяют надежность восприятия. Система, в которой признаки «разрознены», легко сбивается, стоит одному из них дать сбой — например, неожиданная тень или шум приводит к ложному срабатыванию, потому что внутренние признаки не удерживают друг друга. Напротив, модель, где качества сцены хорошо отделены и связаны, поведет себя намного устойчивее: она «поймет», что тень всего лишь изменила освещенность, но не форму и материал объекта, а потому не перестроит гипотезу о наличии объекта. Это критически важно, например, для систем автопилота: редкие или внештатные ситуации (блики, отражения, другие

погодные условия) не должны полностью рушить понимание дороги. Если внутри сети представлены качества “дорога”, “обочина”, “разметка”, “лед/скользкость” и пр., и эти качества согласованно резонируют между собой (например, широкая плоская поверхность + серый материал + горизонтальная ориентация = дорога), то даже при необычном освещении или загрязнении дорожного полотна сеть удержит верную гипотезу и продолжит безопасно вести машину. В исследованиях по автономному вождению все больше внимания уделяется именно таким признакам пригодности и контекстным факторам (“affordances”) наряду с классической детекцией объектов. Концепция аффордансов, введенная психологом Дж. Гибсоном, означает свойства среды в терминах возможных действий (например, поверхность может служить опорой для движения или нет). Для нейросети, освоившей пространство качеств, понятие «проходимости» участка дороги становится таким же осмысленным внутренним параметром, как и класс объекта. Соответственно, модель, сформировавшая ось “проходимость”, будет надежнее различать дорогу, обочину, тротуар и ледяной участок – даже там, где простая сегментационная модель может ошибаться из-за непривычного вида этих объектов. И действительно, практика показала, что богатый опыт разнообразных ситуаций (большие данные) побуждает сети вычленять именно подобные “глубокие” качества: имея множество примеров одного и того же события при разных условиях, модель неизбежно находит, что общего между ними, и использует это для обобщения. Конечный уровень абстракции к которому подходит автономное вождение высокого уровня – понятие «безопасности».

Замечу любопытную параллель с идеями капсульных нейросетей (Capsule Networks), предложенными Г. Хинтоном и соавторами. Капсульный подход как раз пытался добиться устойчивого группирования признаков, вводя механизм «маршрутизации по соглашению» (routing-by-agreement). В CapsNet низкоуровневые «капсулы» (группы нейронов) выдают не скаляр, а вектор – своего рода «позу» или параметры обнаруженного примитива (например, положение и ориентация черты). Эти векторы передаются нескольким возможным капсулам следующего уровня (кандидатам в «родители»). Если несколько дочерних капсул предсказывают сходный результат для родительской (их «позы» согласуются), то связь между ними усиливается, и родительская капсула активируется сильнее. Происходит как бы голосование признаков: когда части соглашаются о целостном объекте, он проявляется. В противном случае (разнобой в предсказаниях) сигнал от части распылится и не повлияет сильно на следующий уровень. Такой процесс явно напоминает и гештальт-группировку по общим трансформациям, и резонанс качеств, о котором говорилось выше. Хотя капсулы не получили широкого приме-

нения из-за вычислительной сложности, их идеи сейчас косвенно воплощаются в трансформерных архитектурах. Механизм внимания фактически позволяет научиться той же маршрутизации по соглашению «routing by agreement», только мягко и распределенно: внимания учатся выделять согласованные группы признаков, не требуя явного алгоритма маршрутизации. Кроме того, трансформеры легко масштабируются и обучаются на данных, тогда как CapsNet были трудны в обучении. Тем не менее, концептуально капсулы внесли ценный вклад: внутренние представления должны уметь учитывать взаимное согласие признаков на счет гипотезы – будь то поза объекта или его материал и прочие качества.

Итоги и перспективы: от устойчивых качеств к динамическому атласу мира

Сведем все вместе. Мы рассмотрели гипотезу о том, что глубокая нейронная сеть выстраивает внутри себя пространство качественных признаков, которые составляют каркас ее понимания окружающего мира. В этом каркасе:

Сверточные слои формируют локальные поля качеств, выделяя базовые различия (градиенты). С их помощью изображение раскладывается на множество “слоев” по типам признаков. Вместо сверточных слоёв могут быть альтернативные способы.

Агрегация и иерархия слоев обеспечивают устойчивость этих признаков к мелким вариациям и поднимают уровень абстракции – вместо сырых пикселей сеть оперирует все более осмысленными свойствами (контуры, формы, текстуры, движения).

Внимание (self-attention) динамически объединяет фрагменты по смысловым связям, вводя синтаксис качеств: выясняется, какие комбинации признаков соответствуют целостным объектам и ситуациям. Это устраняет противоречия и собирает глобальную интерпретацию сцены.

Многомасштабные связи (пирамидальные представления, обходные связи — skip-connections) позволяют согласовывать качества на разных уровнях детализации, от локальных деталей до общего плана сцены. Это как бы завершает формирование внутренней системы координат, “сшивая” ее во всех масштабах.

Резонанс качеств – проявление динамики такой системы: внутренние гипотезы становятся устойчивыми аттракторами. Частично нарушенный ввод или новый ракурс приводит лишь к плавному перемещению точки в пространстве качеств (изменились координаты освещенности, положения и т.п.), но не выбрасывает ее далеко за пределы кластера, соответствующего тому же объекту или событию. Система способна додумывать недостающее, если оно согласуется с прежними качествами, и сдерживать случайные всплески несущественных признаков.

Опираясь на эту основу, можно понять, почему глубокие сети в ряде случаев демонстрируют хорошую переносимость знаний между доменами. Если модель выучила правильные оси – скажем, отделила

форму от текстуры, материал от освещения, – она сможет применить эти факторы и в новых условиях. Например, сеть, привыкшая различать отражение и реальный объект (качество блеск/материал), не перепутает блик на новом, ранее не виденном изображении с отдельным предметом, потому что *в ее внутреннем атласе мира блики – это изменения по оси освещенности, а не по оси материальности*. В результате такая модель будет более устойчивой вне распределения (out-of-distribution): она опирается на глубинные свойства, общие для разных обстановок, вместо того чтобы заучивать частные корреляции обучающей выборки. Практически это подтверждается исследованиями: например, уже упоминавшаяся сеть, обученная с уклоном на форму, оказалась намного надежнее в условиях различных искажений изображений. Подобные наблюдения ведут к пониманию, что путь к устойчивой модели лежит через обучение моделей, вычленяющих правильные факторы вариации.

Возникает вопрос: как нам целенаправленно обучать сети, чтобы у них рождались правильные пространства качеств, а не приходилось полагаться лишь на масштаб данных? Здесь видятся несколько направлений. Одно – уже упомянутая случайность и разнообразие данных (рандомизация домена, агрессивные аугментации), которая автоматически убирает лишние степени свободы и вынуждает модель искать инвариантные оси. Другое – многоцелевое и самоконтролируемое обучение: если ставить перед сетью задачи, требующие предсказывать разные аспекты мира (например, не только класс объекта, но и глубину сцены, и относительное движение, и т.д.), то она гораздо вероятнее разложит представление на осмысленные факторы (это подтверждается успехами самоконтролируемого обучения, где добавление задач типа предсказания положения кусочков изображения или раскраски повышает качество представлений). Третье – архитектурные inductive biases, т.е. заложенные предположения: например, эквариантные слои (капсулы, групповая свертка) уже направляют модель к выделению позы отдельно от объекта, а рекуррентные модули типа LSTM или фильтров Калмана встраивают идею устойчивости во времени. Все это – инструменты, которые можно рассматривать как средства “настройки” внутреннего пространства модели.

Наконец, взглянем на горизонт: к чему приведет развитие идей качественного пространства? Я полагаю, что мы движемся к созданию у машин своеобразного «динамического атласа мира». Под этим образным термином подразумевается внутреннее представление, где координатами служат фундаментальные качества внешнего мира, а динамика системы определяется резонансами (правилами совместного изменения) этих координат. Такой атлас будет постоянно обновляться по мере поступления новых наблюдений, но обновляться вдоль понятных осей.

В сущности, это и есть мечта о мировой модели (world model) в искусственном интеллекте: система, которая обладает встроенным представлением об устройстве окружающей среды и способна симулировать последствия своих действий или внешних событий. Исследования в этом направлении уже ведутся. Например, в работе “*World Models*” (Д. Ха, Ю. Шмидхубер, 2018) агента научили строить сжатое латентное представление окружения и генерировать в этом пространстве прогнозы будущих кадров – по сути, обучили внутренний симулятор. В проектах DeepMind по генеративным моделям восприятия (*Generative Query Network*, 2018) машина училась, посмотрев на несколько изображений сцены, представлять ее так, чтобы потом сгенерировать вид под новым углом. В результате сеть явно выучивала внутри факторы типа “расположение объектов”, “цвета”, “освещение”, и могла перестраивать изображения при изменении этих факторов. Это и есть прототип динамического атласа: модель знает, что если источник света переместился, то тени сдвинутся (ось освещенности изменилась), но форма объектов осталась той же; или если один объект закрывает другой, то при движении камеры скрытый объект может появиться (ось ракурса влияет на видимость).

Подобные системы совмещают восприятие и генерацию, как мы говорили вначале, в единый процесс построения и обновления гипотез. Не важно, идет ли речь об изображении, звуке, радиоволне или текстовом описании – везде машина должна извлечь устойчивые смыслы (качества) и уметь оперировать ими. Отличный пример – большие языковые модели: они ведь тоже строят вероятностное пространство смысловых факторов языка и генерируют новые фразы на основе обучения предсказанию пропущенных частей. По сути, GPT и подобные модели формируют внутри себя пространство понятий, близкое к тому, как человек организует знания. Интеграция таких моделей с визуальными (как в CLIP, DALL-E и др.) позволяет надеяться, что в будущем мы получим универсальные мультимодальные представления, где качества мира из разных ощущений объединяются (зрение дает форму и цвет, звук – материал и полость пространства, текст – символические метки и абстрактные связи). И далее машинное восприятие выйдет далеко за пределы человеческого восприятия за счет интеграции всевозможных датчиков, спектроанализа, телеметрии и прочего. Машины смогут рассказать нам больше об окружающей нас действительности при экстраполяции видимого нами мира с его законами резонанса на скрытые от наших органов чувств области физического мира.

Современная тенденция нейрокомпьютинга – переход от поверхностного распознавания к глубокому, резонансному пониманию – требует переосмысления внутренних представлений в терминах «пространства качеств». Гипотеза, изложенная в этой книге, призвана служить шагом в этом направлении. Она связывает воедино ряд разрозненных

идей: иерархию признаков, механизм внимания, принцип анализа-через-синтез, аффордансы, устойчивые аттракторы – под общей концепцией внутреннего пространства, где объект задается конфигурацией устойчивых качественных координат. Эта точка зрения не противоречит известным результатам, а напротив, позволяет их объяснить единообразно. Более того, она дает намек, куда двигаться дальше в архитектурах и методах обучения. Если мы хотим, чтобы машина лучше строила гипотезы, переносила знания, объясняла свои решения, нам нужно, чтобы ее внутренний мир был достаточно абстрактен и разложен на осмысленные величины. Это фактически приближает нас к символическому, человеческому уровню понимания, но не жертвуя сложностью и богатством, которые дают нейросетевые представления. Впереди большая работа: как именно внедрить требования к устойчивости и предсказуемости в функцию потерь, как проверить наличие «качественных осей» внутри черного ящика сети, как организовать взаимодействие резонансных модулей в большой системе. Тем не менее, уже сейчас прослеживается, что именно умение выделять и использовать базовые качества реальности может стать тем ключевым мериллом, по которому будут оцениваться интеллектуальные системы нового поколения – подобно тому, как в прошлом десятилетии их оценивали по точности распознавания образов.

Понятие «пространства качеств» созвучно с теорией *conceptual spaces* в когнитивной науке (П. Гардэнфорс и др.), где концепты представляются как области в многомерных пространствах признаков. В нашем случае нейросеть самостоятельно формирует подобное пространство в процессе обучения, обнаруживая «природные» оси вариаций. Также стоит отметить параллели с подходами к развязке факторов (disentanglement), прогнозирующему кодированию (predictive coding) и теориями восприятия в нейронауке, рассматривающими мозг как систему, постоянно предсказывающую сенсорный вход. Все эти направления указывают на важность поиска правильных внутренних представлений – таких, где изменения в одном измерении отражают настоящие независимые изменения во внешнем мире. Гипотеза качественного пространства в машинном зрении – еще один шаг на пути к этой цели, соединяющий практические находки глубокого обучения с фундаментальным вопросом: *что же на самом деле видит и понимает машина, когда смотрит на мир?*

Очень показательно, что похожая логика начинает проявляться не только в машинном зрении, но и в нейронауке. В 2026 году была опубликована работа «Мета-обучение в контексте обеспечивает межсубъектное декодирование мозга без дообучения» (Meta-learning In-Context Enables Training-Free Cross Subject Brain Decoding), где авторы прямо ставят проблему индивидуальной вариативности мозга. Один человек и другой человек могут видеть один и тот же зрительный стимул, но его

след в данных функциональной магнитно-резонансной томографии будет расположен по-разному: отличаются анатомия, функциональная организация коры, количество доступных вокселей, параметры сканера, шум и протокол записи. Раньше это часто вынуждало строить отдельную модель под каждого человека или дообучать систему на данных нового субъекта. Авторы предлагают другой путь: модель не привязывается к жёсткой карте мозга, а учится быстро выводить, как именно данный мозг кодирует зрительную информацию, по небольшому набору пар «изображение — мозговая активность». В статье подчёркивается, что подход обобщается на новых людей без дообучения, не требует анатомического выравнивания и не требует совпадения стимулов между испытуемыми.

Технически это устроено как двухступенчатая инверсия кодирующей модели. Сначала система для каждого вокселя — то есть маленького объёмного элемента измерения в данных магнитно-резонансной томографии — по контексту из нескольких изображений и ответов мозга оценивает его индивидуальную функцию отклика. Важно, что речь идёт не о «нейроне» в буквальном смысле: воксель объединяет активность множества нейронов и кровотоковый ответ в небольшой области ткани. Затем вторая часть модели собирает множество таких воксельных описаний и текущих активаций в общий контекст и пытается восстановить не само изображение в пикселях, а его внутреннее зрительно-семантическое представление — эмбединг изображения. Иными словами, система сначала понимает, какие зрительные признаки кодирует каждый измеряемый участок мозга у конкретного человека, а затем собирает из этих участков общее объяснение увиденного.

В этой работе не делается простая свёртка по анатомической карте мозга. Каждый воксель представлен как элемент контекста со своими оценёнными параметрами кодирования и текущей активацией, а дальнейшее объединение выполняется трансформерной моделью. В этом смысле работа даёт очень сильную параллель к гипотезе качественного пространства. Модель не учится распознавать «воксель номер такой-то у человека номер такой-то». Она учится более абстрактной процедуре: как по небольшому контексту понять, какие внутренние зрительные качества кодирует данный мозг, а затем инвертировать эту индивидуальную систему кодирования обратно в пространство зрительных представлений. То есть между людьми переносится не конкретная карта активаций, при показе изображения, а более общий принцип связи между стимулом, функцией отклика и скрытым смысловым представлением этого изображения. Именно это и похоже на переход между доменами: каждый человек становится отдельным доменом со своей геометрией, шумами и координатами, но под ними обнаруживается общий слой

закономерностей — правила, по которым зрительная система кодирует формы, сцены, категории и отношения.

Для нашей темы это особенно важно потому, что здесь абстрактное обобщение показано не на дорожных сценах, не на бильярдных шарах и не на смене камер, а на самом сложном носителе — на разных человеческих мозгах. Если система может адаптироваться к новому человеку по малому числу примеров и при этом не требует жёсткого совпадения анатомии, значит, она действительно работает не с поверхностной координатной сеткой, а с более высоким функциональным описанием. Это почти прямой пример того, что я называю качественной гипотезой: устойчивое представление должно сохранять смысл при смене внешней реализации.

Эту работу можно рассматривать как важный аргумент в пользу будущей универсальности внутренних представлений. Она показывает, что сильная модель должна не только классифицировать входной сигнал, но и быстро строить субъектно-специфичную карту соответствий между наблюдаемыми данными и скрытым смысловым пространством. Это уже не просто распознавание, а адаптивное понимание кодирования. Если перенести эту мысль обратно в машинное зрение, то задача будущих архитектур будет состоять не в том, чтобы один раз выучить идеальную сетку признаков, а в том, чтобы уметь быстро находить, как конкретная камера, конкретная сцена, конкретный человек или конкретный мозг проецирует реальность в свои данные. Именно здесь понятие «пространства качеств» становится особенно полезным: оно даёт язык для описания того общего слоя, который сохраняется при переходе от одного домена к другому.

Когда я оглядываюсь на весь путь — от первых детекторов признаков до сегодняшних моделей, которые строят сцены, держат треки, выдерживают смены домена и умеют учиться сами почти без разметки, — я все чаще ловлю себя на том, что машинное зрение перестало быть про анализ видео. Оно стало про то, как вообще возникает устойчивый смысл из сырого потока ощущений. Мы начали как инженеры: хотели, чтобы система находила объект, измеряла траекторию, не путалась в бликах и тенях, не ломалась на дождливом асфальте. Мы измеряли качество по метрикам, по задержке, по стабильности. Но чем выше становилась планка, тем яснее становился главный факт: устойчивость не покупается отдельно. Ее нельзя прикрутить как модуль. Она вырастает только из внутренней организации представления — из того, как модель устроила мир в себе.

И вот тут в какой-то момент все, что казалось сугубо техническим, начинает звучать неожиданно по-человечески. Мы говорили «инвариантность», а это по сути способность не терять смысл при изменении

формы ощущения. Мы говорили «эквивариантность», а это способность переносить смысл вместе с движением и симметрией. Мы говорили «временная согласованность», а это почти буквальная память: удержание идентичности. Мы говорили «внимание как динамический граф», а это механизм выбора того, что с чем связано, что чему противоречит. Мы говорили «резонансная совместимость качеств», а на человеческом языке это звучит так: смысл рождается не из отдельного признака, а из взаимного подтверждения, из того, что разные стороны опыта складываются в устойчивую конфигурацию.

И тогда качественная гипотеза — в самом мягком, не радикальном варианте — становится попыткой назвать то, что мы постоянно видели на практике, но описывали чужими словами. Что внутренняя реальность модели — это поле качеств: непрерывных, многомасштабных, связанных, иногда конфликтующих, иногда усиливающих друг друга. И что восприятие — это не акт распознавания, а акт нахождения устойчивой конфигурации в этом поле, которая одновременно объясняет наблюдение и не разрушает внутренние связи.

Вроде бы мы говорим про нейросети. Но становится трудно не заметить, насколько это похоже на нас самих. Человеческое восприятие ведь тоже не сводится к «детектированию объектов». Мы не видим мир как набор подписанных вещей и траекторий. Мы видим его как набор вероятностей и запретов, как ткань напряжений и разрешений: можно пройти, нельзя пройти, близко, далеко, опасно, знакомо, неясно, притягивает внимание, ускользает. Мы узнаем объект не потому, что каждый пиксель совпал, а потому, что множество качеств сошлось в устойчивый образ: форма, движение, контекст, память, ожидание следующего мгновения. И если внутри все сошлось, мы чувствуем уверенность — не как число, а как состояние (резонанс).

То есть, по сути, мы тоже живем в качественном пространстве. Просто называем его иначе. Если так, то становится возможной совсем другая концовка книги о машинном зрении. Не про то, что будет после следующего архитектурного скачка, а про то, что машинное зрение, как дисциплина, неожиданно приблизила нас к вопросу о природе смысла. Причем не в философском смысле «что такое истина», а в очень практическом: что именно должно совпадать, чтобы сохранялся смысл опыта? Какие инварианты делают мир «тем же самым», несмотря на смену точки зрения, освещения, состояния? Какая структура связей делает наше восприятие устойчивым? Почему одни ошибки в восприятии мы сглаживаем, а на других «спотыкаемся»? И почему разные «правила совместимости» формируют разные картины мира в нашем восприятии?

Если продолжать эту мысль, она подводит к самому интересному: машинное зрение может стать не только инструментом для распознавания мира, но и инструментом для понимания того, как мир вообще может быть распознаваем. Не зеркалом реальности, а зеркалом механизмов, которые превращают реальность в смысл. Возможно, именно в этом и состоит дальняя перспектива всей нашей технологической эпохи: мы строим машины, которые учатся видеть, и, наблюдая за тем, как они это делают, учимся заново видеть себя.

Иногда мне кажется, что качественная гипотеза — это осторожный намек на то, что мы нащупываем универсальный язык описания устойчивых миров. Не только сцен видео, а вообще состояний, которые могут быть удержаны, воспроизведены и продолжены во времени. Если резонанс — это способность качеств поддерживать друг друга, то смысл — это локальный резонанс в океане возможного. Если информация — это ограничение, которое сужает хаос до траектории, то восприятие — это выбор траектории из множества, который делает будущее предсказуемым. Если энергия — это цена удержания качественной конфигурации, то прикладывая усилия по изменению своей системы восприятия, вероятно, можно менять интерпретацию окружающей действительности.

И вот тут я позволю себе совершенно спекулятивный поворот. Его невозможно проверить сегодня, и я не предлагаю его как научный факт. Я предлагаю его как метафору-ориентир, которая неожиданно хорошо стыкуется с тем, что мы уже обсудили.

А что, если мир на самом фундаментальном уровне действительно есть конфигурация качеств? Не в смысле признаков нейросети, а в смысле первичных степеней свободы, которые могут входить в резонанс и образовывать устойчивые режимы. Физики когда-то пытались говорить об этом через поля, потом через кванты, потом через струны и их моды колебаний. И если отбросить исторический багаж слов, останется простая идея: то, что существует, существует потому, что способно удерживаться. А удерживаться способно то, что попадает в устойчивые резонансные замыкания. Все остальное — шум, виртуальная возможность, альтернативная ветка реальности. Физическая способность нашего восприятия удерживаться именно в такой ветке мироздания возможна потому, что находит с ним общее качественное соответствие.

Тогда становится соблазнительно представить, что «параллельные вселенные», — если они вообще имеют смысл, — могут различаться как соседние энергетические минимумы в гигантском ландшафте качественных конфигураций. Как разные устойчивые траектории в пространстве возможностей. Как разные сценарии, которые отличаются локальными развилками — не потому, что кто-то сделал случайный выбор,

а потому, что в многомерном мире устойчивостей существует множество близких минимумов, и мыслящая система, которая состоит из той же ткани качеств, может усилием воли стечь в один или в другой в зависимости от своего качественного резонанса с осями этого ландшафта.

Это пока невозможно проверить. Но эта мысль меня удивительно вдохновляет: она говорит, что реальность — это не одна ось времени, а ландшафт, и мы живем в тех долинах и видим все вокруг себя именно так, как замыкается наиболее вероятная динамика при нашей текущей качественной совместимости. В этой метафоре судьба, причинность, вероятность перестают быть случайностью и становятся геометрией ландшафта. А качество восприятия — это навигация по нему, перебор резонансов, поиск устойчивой интерпретации среди множества возможных в прямом и переносном смысле.

Сегодня в машинном зрении мы учимся строить системы, которые не просто классифицируют, а удерживают устойчивые представления в изменчивом мире. Мы учимся кодировать законы совместимости. Мы учимся выращивать внутренние ландшафты и кодировать реальность в вероятностном хаосе триллионов параметров. И в этом смысле развитие машинного зрения — это развитие языка, на котором мир может быть описан.

Машинное зрение начиналось как попытка научить машину анализировать данные. Но, возможно, в пределе это превращается в попытку научить машину — и нас вместе с ней — находить более подходящий смысл: находить такие конфигурации качеств, которые не распадаются при перемене условий, которые подтверждаются временем, которые вступают в резонанс с другими частями опыта, которые способны продолжаться. И если это правда, то качественная гипотеза — это про то, как мы постепенно приближаемся к языку, который связывает восприятие, знание и реальность в одну структуру. Если мир действительно устроен как ландшафт устойчивых качественных конфигураций, то развитие машинного зрения — это не только путь к более умным машинам. Это путь к более точному пониманию того, что значит быть наблюдателем в этом мире. И, возможно, к более точному пониманию того, что значит быть Человеком.

Благодарности

Хотел бы выразить искреннюю благодарность людям и организациям, которые причастны к написанию этой книги в самом широком смысле и без которых она бы не состоялась. Поскольку книга во многом и про мой практический опыт системного применения машинного зрения в спорте, то считаю уместным упомянуть тут и людей причастных к его созданию и развитию, а также всех причастных к спортивной дисциплине, к которой он относится. Перечислю их поименно и отмечу вклад каждого из них. Благодарю моего отца - Юрия Владимировича Садовника, за то, что он привёл в спорт и привил интерес к бильярду и на своём примере передал инженерный склад ума. Благодарю Алексея Борисовича Паяркова, видного бильярдиста, предпринимателя, за культивирование и развитие бильярдного спорта среди молодежи в моем городе и за его пределами в начале 2000х. Благодарю Кирила Жиздюка за развитие бильярдного спорта среди молодежи, продолжающего эстафету Алексея Борисовича. Благодарю Владимира Марценюка — визионера и моего делового партнера, за поддержку во всех моих инновационных начинаниях. Именно ему я обязан всеми своими достижениями.

Благодарю Екатерину Корниенко — верного соратника, который с самого начала проекта помогает мне в его реализации - готовит датасеты, тестирует и всё проверяет, развивает онлайн сообщество. Благодарю Виктора Горбаня — программиста полного цикла, прошедшего под моим руководством от выпускника ВУЗа до опытного разработчика, за его терпение и точное выполнение всех моих замыслов. Благодарю Ваню Волкова — программиста C++ за помощь и вклад в развитие проекта. Благодарю Георгия Маруха — программиста, математика за вклад в развитие проекта и подготовку теоретической базы по оценке сложности ударов. Благодарю Максима Петрова, за системное администрирование. Благодарю Сергея Лохматова — друга детства, за то, что помог сформулировать общую идею проекта. Юру Попова - спортивного товарища, партнера по команде, за поддержку в проекте и совместные победы. Дмитрия Угай - спортивного товарища за поддержку в проекте. Благодарю Федора Горта, Руслана Чинахова, Константина Золотилова, Андрея Сероштаня, Михаила Каблукова — видных российских бильярдистов за положительные отзывы, обратную связь и поддержку в работе над проектом. Владимира Алексеевича Куликова - председателя регионального общества Изобретателей и Рационализаторов за поддержку в проекте. Александра Ивановича Черноусенко за поддержку в проекте и личный пример. Фонд Со-

действия Инновациям, Фонд Сколково, Редактора, Верстальщика, Издателя. Сергея Игоревича Баязитова благодарю за поддержку и помощь в продвижении.