МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра информационных систем управления

КОЛБ Олег Олегович

МЕТОДЫ И АЛГОРИТМЫ ОБНАРУЖЕНИЯ ОБЪЕКТОВ НА ОСНОВЕ СЕМЕЙСТВА АРХИТЕКТУР YOLO

Магистерская диссертация

Научный руководитель
М. М. Лукашевич
кандидат технических наук, доцент

Допущена к защите
«____»_____ 20__ г.
Заведующий кафедрой информационных
систем управления
доктор технических наук, доцент
А. М. Недзьведь

Минск, 2025

Оглавление

ВВЕДЕНИЕ

Современное общество всё глубже проникает в цифровое пространство, где обработка визуальной информации стала неотъемлемой частью как научных изысканий, так и индустриальных решений. Обнаружение объектов на изображениях — ключевая задача в области компьютерного зрения, лежащая в основе таких приложений, как автопилотируемые системы, медицинская диагностика, контроль качества производства, мониторинг инфраструктур и безопасности.

Исторически развитие методов детекции шло от медленных двухстадийных моделей к стремительно работающим одностадийным, среди которых архитектура YOLO (You Only Look Once) заняла особое место. Однако классические методы локализации ограничены прямоугольными рамками, не способными адекватно описывать наклонные или вытянутые объекты. Это особенно критично в таких задачах, как анализ микросхем, судов или медицинских изображений, где объекты часто ориентированы под произвольными углами.

С учётом этого в последние годы наблюдается бурный рост интереса к ориентированным ограничительным рамкам (OBB), которые позволяют существенно повысить точность локализации, особенно в плотных и сложных сценах. Новейшие модификации YOLO, включая версии v8, v11, v12, поддерживают OBB и предоставляют уникальные возможности для построения универсальных, но точных моделей детекции.

Таким образом, исследование эффективности современных моделей YOLO в задачах с OBB и формализация сравнительного анализа на различных прикладных датасетах представляют собой актуальную научную и практическую проблему, на которую данная работа даёт систематизированный и экспериментально подтверждённый ответ.

Диссертация состоит из пяти глав, логически выстроенных от теоретического фундамента к практическим результатам:

- **Глава 1** содержит введение в задачу детекции объектов и исторический контекст развития подходов.
- **Глава 2** представляет обзор методов обнаружения объектов, с акцентом на RCNN и семейство YOLO.
- **Глава 3** посвящена подробному анализу архитектур YOLO, включая модификации с поддержкой ориентированных рамок.
- **Глава 4** рассматривает форматы аннотаций, метрики качества, особенности подготовки данных и специфику OBB.
- **Глава 5** содержит описание экспериментального окружения, анализ датасетов, результаты обучения моделей и их сравнительный разбор.

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Целью магистерской диссертации является сравнительный анализ и практическая апробация моделей YOLO с поддержкой OBB на изображениях из различных прикладных областей — промышленной визуализации, аэрофотосъёмки и медицины. Исследование сосредоточено на оценке производительности, точности и способности моделей к обобщению в зависимости от структуры сцены.

В рамках работы поставлены следующие **задачи**: изучение теоретической базы; анализ эволюции архитектур YOLO до версии YOLOv12; изучение форматов аннотаций и метрик качества (IoU, mAP, Precision, Recall); подготовка и анализ трёх прикладных датасетов; обучение моделей различных конфигураций; проведение сравнительного анализа и визуализация результатов.

Объектом исследования являются алгоритмы и модели глубокого обучения, применяемые в задачах обнаружения объектов на изображениях.

Предмет исследования — архитектуры YOLOv8, YOLOv11, YOLOv12 с поддержкой OBB, их обучение, оценка и практическое применение.

Методы исследования включают: анализ литературы, методы машинного обучения с использованием библиотек Ultralytics и scikit-learn, визуализацию результатов, количественную оценку моделей и анализ ошибок.

Научная новизна и результаты работы. Проведено системное сравнение трёх поколений моделей YOLO с поддержкой OBB-разметки на разнообразных по структуре и сложности датасетах. Выявлены зависимости между размером модели и типом сцены, даны рекомендации по выбору архитектуры. Продемонстрирована состоятельность моделей YOLOv12 даже при отсутствии предобученных весов. Подготовлены мета-таблицы и визуализации, дающие всестороннее представление о качестве работы каждой модели.

Результаты апробированы на научных семинарах кафедры и в рамках подготовки тезиса для II Международной научно-практической конференции (TMMIE-II). Разработанные модели и методики пригодны для воспроизводимого тестирования и дальнейшего применения в прикладных системах

Магистерская диссертация состоит из оглавления, введения, общей характеристики работы, пяти глав, заключения, списка использованных источников и приложений. В работе представлено 2 приложения, 43 рисунка. Общий объем диссертации составляет 99 страниц. Список использованных источников занимает 4 страницы и включает 47 наименований на английском языке.

АГУЛЬНАЯ ХАРАКТАРЫСТЫКА ПРАЦЫ

Мэтай магістарскай дысертацыі з'яўляецца параўнальны аналіз і практычная апрабацыя мадэляў YOLO з падтрымкай OBB на выявах з розных прыкладных абласцей — прамысловай візуалізацыі, аэрафотаздымкі і медыцыны. Даследаванне засяроджана на адзнацы прадукцыйнасці, дакладнасці і здольнасці мадэляў да абагульнення ў залежнасці ад структуры сцэны.

У рамках работы пастаўлены наступныя **задачы**: вывучэнне тэарэтычнай базы; аналіз эвалюцыі архітэктур YOLO да версіі YOLOv12; вывучэнне фарматаў анатацый і метрык якасці (IoU, mAP, Precision, Recall); падрыхтоўка і аналіз трох прыкладных датасетаў; навучанне мадэляў розных канфігурацый; правядзенне параўнальнага аналізу і візуалізацыя вынікаў.

Аб'ектам даследавання з'яўляюцца алгарытмы і мадэлі глыбокага навучання, якія прымяняюцца ў задачах выяўлення аб'ектаў на малюнках.

Прадмет даследавання - архітэктуры YOLOv8, YOLOv11, YOLOv12 з падтрымкай OBB, іх навучанне, ацэнка і практычнае прымяненне.

Метадалагічная база ўключае: аналіз літаратуры, метады машыннага навучання з выкарыстаннем бібліятэк Ultralytics і scikit-learn, візуалізацыю вынікаў, колькасную адзнаку мадэляў і аналіз памылак.

Навуковая навізна і вынікі працы. Праведзена сістэмнае параўнанне трох пакаленняў мадэляў YOLO з падтрымкай OBB-разметкі на разнастайных па структуры і складанасці датасетах. Выяўлены залежнасці паміж памерам мадэлі і тыпам сцэны, дадзены рэкамендацыі па выбары архітэктуры. Прадэманстравана заможнасць мадэляў YOLOv12 нават пры адсутнасці прадугледжаных вагаў. Падрыхтаваны мета-табліцы і візуалізацыі, якія даюць ўсебаковае ўяўленне аб якасці працы кожнай мадэлі.

Вынікі апрабаваны на навуковых семінарах кафедры і ў рамках падрыхтоўкі тэзіса для II Міжнароднай навукова-практычнай канферэнцыі (TMMIE-II). Распрацаваныя мадэлі і методыкі прыдатныя для прайграванага тэсціравання і далейшага прымянення ў прыкладных сістэмах

Магістарская дысертацыя складаецца з зместа, увядзення, агульнай характарыстыкі працы, пяці кіраўнікоў, заключэння, спісу выкарыстаных крыніц і прыкладанняў. У рабоце прадстаўлена 2 дадаткі, 43 малюнкі. Агульны аб'ём дысертацыі складае 99 старонак. Спіс выкарыстаных крыніц займае 4 старонкі і ўключае 47 найменняў на англійскай мове.

GENERAL CHARACTERISTICS OF THE THESIS

The **aim** of this master thesis is the comparative analysis and practical validation of OBB-enabled YOLO models on images from different application domains - industrial imaging, aerial photography and medicine. The research focuses on evaluating the performance, accuracy and generalisation ability of the models depending on the scene structure.

The **objectives** of the work are: to study the theoretical background; to analyse the evolution of YOLO architectures up to YOLOv12; to study annotation formats and quality metrics (IoU, mAP, Precision, Recall); to prepare and analyse three application datasets; to train models of different configurations; to perform comparative analysis and visualise the results.

The **object** of the study is deep learning algorithms and models used in the tasks of object detection in images.

The **subject** of the study is YOLOv8, YOLOv11, YOLOv12 architectures with OBB support, their training, evaluation and practical application.

The **methodological framework** includes: literature review, machine learning methods using Ultralytics and scikit-learn libraries, visualisation of results, quantification of models and error analysis.

Scientific novelty and results of the work. A systematic comparison of three generations of YOLO models with OBB markup support on datasets of different structure and complexity is carried out. The dependencies between model size and scene type are revealed, and recommendations on architecture selection are given. Demonstrated the consistency of YOLOv12 models even in the absence of pre-trained weights. Meta-tables and visualisations are prepared, giving a comprehensive view of the performance quality of each model.

The results were tested at scientific seminars of the department and in the framework of preparation of the abstract for the II International Scientific and Practical Conference (TMMIE-II). The developed models and methods are suitable for reproducible testing and further application in applied systems

Master's thesis consists of a table of contents, introduction, general characterisation of the work, five chapters, conclusion, list of used sources and appendices. The work contains 2 appendices, 43 figures. The total volume of the thesis is 99 pages. The list of used sources occupies 4 pages and includes 47 titles in English.

ГЛАВА 1. АППАРАТ НЕЙРОСЕТЕЙ В ЗАДАЧАХ КОМПЬЮТЕРНОГО ЗРЕНИЯ

1.1 Подходы к решению задач компьютерного зрения

Размышления о создании механизма, способного мыслить и действовать подобно человеку, появились не в лабораториях Кремниевой долины, а в тени колоннад афинской школы. Ещё в античности философы поднимали вопрос о возможности переноса рациональности в неорганическое тело — не как инженерную задачу, а как продолжение спора о природе души. Эти идеи не имели ни кода, ни меди, но уже тогда в них звучал набросок будущего алгоритма.

Спустя столетия, в 1842 году, Ада Лавлейс — комментируя работу Чарльза Бэббиджа — предположила, что вычислительная машина может выполнять не только числовые операции, но и логические конструкции, если те будут заданы в виде инструкций. Это утверждение, как бы между строк, стало первым фундаментом концепции универсального интеллекта, пусть и реализуемого через предельно ограниченные механизмы своего времени [1].

С появлением цифровых машин в середине XX века идея создания искусственного разума обрела конкретные очертания. Возникла новая область — искусственный интеллект, в рамках которой ставились амбициозные задачи: распознавать речь, интерпретировать изображения, принимать решения в непредсказуемых условиях. Однако быстро выяснилось, что те задачи, которые человек решает почти автоматически, для машины превращаются в непреодолимую преграду. Простая визуальная сцена, понятная ребёнку, становилась для программы хаотичным массивом пикселей без смысла.

Первые попытки приблизить машину к интеллектуальному поведению строились на жёсткой логике: человек вручную формализовывал знания о мире — описывал закономерности, создавал иерархии понятий, прописывал условия. Компьютер не обучался, он исполнял. Если ситуация выходила за пределы предусмотренного — система оказывалась беспомощной. Такой подход оказался нежизнеспособным в открытых, изменчивых условиях. Он требовал не только колоссального труда, но и предполагал невозможность полного охвата реальности правилами.

Качественный сдвиг произошёл с появлением многоуровневых архитектур, где каждый слой анализировал данные предыдущего, выделяя всё более абстрактные признаки. Это позволило отказаться от ручного программирования логики и передать процесс обобщения самой системе, обучаемой на примерах. Так возникло глубокое обучение — не как модный

термин, а как парадигма: модель не знает, как устроен мир, но она учится понимать его через повторяющийся опыт, через паттерны, которые она извлекает из сырых данных.

В рамках этой новой логики быстро стало ясно, что компьютерное зрение — не просто частный случай ИИ, а его фронтир. Здесь, на стыке биологии, информатики и когнитивистики, развернулась борьба за возможность автоматически интерпретировать визуальную информацию. Речь идёт не только о том, чтобы «увидеть» — а о том, чтобы понять, что изображено на экране, как это соотносится с контекстом и какие выводы из этого следуют.

За последние годы развитие этого направления ускорилось — и не только благодаря теории. Выросла доступность вычислительных ресурсов, были собраны масштабные датасеты, улучшились архитектуры сетей. Однако по-прежнему сохраняется двоичность подходов: с одной стороны — классические методы компьютерного зрения, основанные на правилах, фильтрах и геометрических признаках; с другой — модели глубокого обучения, которые не ищут правила, а обучаются распознавать закономерности, даже если они не поддаются формализации. история методов обнаружения объектов в компьютерном зрении напоминает постепенное раскрытие сложной хронологической карты (рисунок 1.1), где каждая новая точка — это не просто очередной алгоритм, а концептуальный сдвиг, меняющий представление о том, как машина может "видеть" [2; 3].

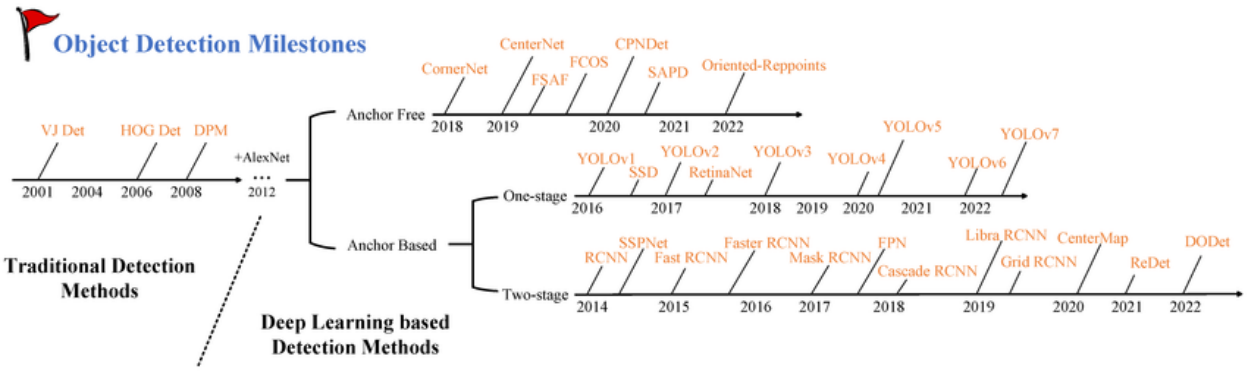

Рисунок 1.1 — Историческое развитие детекции объектов

1.2 Задачи компьютерного зрения

Повторяясь, интерпретировать изображение — значит вычленить смысл из набора пикселей. Для машины это не просто процесс обработки — это имитация человеческого восприятия, формализованная через математику, архитектуру и данные. Задачи компьютерного зрения лежат на этом стыке: от простых — как определить, что на картинке изображён кот, до сложных — как очертить границы каждой его лапы, и понять, в каком положении он прыгает.

Ключевыми направлениями здесь являются — классификация, сегментация (семантическая, экземплярная, паноптическая), детекция и обнаружение ключевых точек (рисунок 1.2). Каждое из них — как отдельный орган чувств: классификация отвечает за узнавание, сегментация — за пространственное восприятие, ключевые точки — за скелетную структуру, а детекция объединяет всё это в осмысленную локализацию [4; 5].

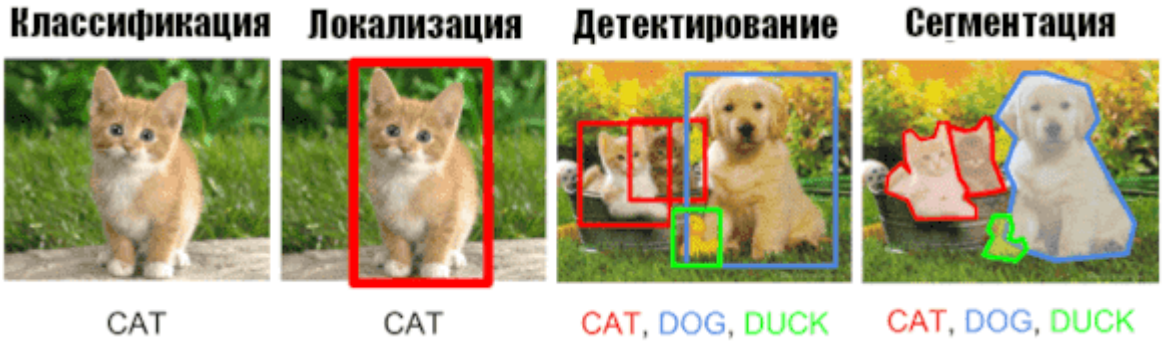

Рисунок 1.2 — Пример задач компьютерного зрения

Прежде чем углубиться в детекцию объектов, остановимся на базовых задачах, без которых невозможен переход к более сложным решениям.

1.2.1 Классификация объектов

Классификация — это, по сути, сжатие опыта. На входе — визуальный мир во всей его сложности. На выходе — одно слово: «собака», «самолёт», «фотоаппарат». Алгоритм проходит путь от сырых данных к обобщённому решению. Он не ищет формы — он распознаёт суть.

С технической точки зрения задача классификации решается путём построения распределения вероятностей по классам. Модель (чаще всего CNN) выдаёт наиболее вероятный вариант:

$$\hat{y} = \underset{y}{argmax}\, P(y \mid image)$$

Если бы весь процесс был интуитивно понятен, не случилось бы 2012 года. Именно тогда AlexNet, свёрточная нейросеть, обученная на ImageNet (1,2 миллиона размеченных изображений, 1000 классов), совершила переворот: уровень ошибки по сравнению с предыдущими методами снизился почти вдвое. Это не была просто инженерная победа. Это было заявление: «машина может видеть лучше, если научить её правильно» [6].

С тех пор архитектуры стали глубже, шире, умнее: VGG, GoogLeNet, ResNet и др. Некоторые из них превосходили среднестатистического человека в точности на тестах. Но что важно — они не просто распознавали изображения. Они стали базисом для всех последующих задач: детекции,

сегментации, captioning, VQA (Visual Question Answering). Без уверенной классификации — нет уверенного зрения [9].

Сегодня задача классификации как самостоятельное направление решена для большинства «закрытых» категорий. Но в открытых системах (например, в реальном мире, где объекты могут быть новыми) она всё ещё требует тонкой настройки — в том числе через transfer learning и few-shot методы.

Классификация — это не цель, это инструмент. Она как имя, присвоенное незнакомцу: сначала узнаём, потом начинаем общаться. Машина делает то же самое — по-своему, через слои, фильтры и градиенты. Но результат один — узнавание.

1.2.2 Сегментация изображения

Если классификация — это искусство называть, то **сегментация** — это наука очерчивать. В компьютерном зрении сегментация выполняет почти хирургическую работу: она расщепляет изображение до уровня каждого пикселя, распределяя его не просто по цвету или яркости, а по смыслу. Эта операция не грубая — она точная. Задача сегментации — не только сказать, что изображено, но где именно это находится и что с чем связано.

Сегментация — не монотонная дисциплина. Она существует в трёх различных уникальных направлениях, каждое из которых отвечает на разные вопросы.

Семантическая сегментация (semantic segmentation). Воображаем, что перед нами — уличная сцена: небо, тротуар, проезжая часть, машины и пешеходы. Семантическая сегментация стремится к целостной интерпретации: каждый пиксель окрашивается в цвет своего класса. Все пиксели неба — синие, все пиксели автомобилей — красные, люди — зелёные. Алгоритм не различает, один это человек или их трое. Он просто знает: «это — человек». Модель видит массы, но не личности. Для неё не существует различия между человеком А и человеком Б. Они оба — просто человек (рисунок 1.3).

Сегментация экземпляров (instance segmentation). Теперь картина становится живее. Те же трое пешеходов, но теперь модель говорит: «вот человек 1, вот человек 2, а это — человек 3». Это уже не просто семантика, а индивидуальность. Здесь начинает работать Mask R-CNN — один из первых флагманов этой парадигмы.

Каждому объекту присваивается не только класс, но и маска — как личный силуэт. Получается не просто раскраска, а визуальное досье: где

именно находится каждый конкретный объект, даже если их классы совпадают.

Это особенно важно, когда объекты перекрываются. Там, где семантика рисует общую массу, экземплярная сегментация — рисует границы, различия, индивидуальность (рисунок 1.3).

Паноптическая сегментация (panoptic segmentation). Паноптическая сегментация — это объединение двух предыдущих подходов под единый знаменатель. Название говорит само за себя: паноптика, то есть «видеть всё».

В этой задаче модель должна точно различать, какие пиксели принадлежат «вещам» — исчисляемым объектам вроде человека, машины, дерева. И какие — «веществам»: туман, дорога, небо. Первые требуют индивидуальности, вторые — общего смысла.

Представьте идеальную разметку сцены: «человек 1», «человек 2», «машина 3», «небо», «здание», «дорога». Ни одного пикселя без ярлыка. Ни одного перекрытия. Это — полное описание мира глазами машины.

Именно эту концепцию в 2019 году предложили Kirillov, A и соавторы [8], введя строгую метрику — PQ (Panoptic Quality), объединяющую точность разметки и способность к дифференциации (рисунок 1.3).

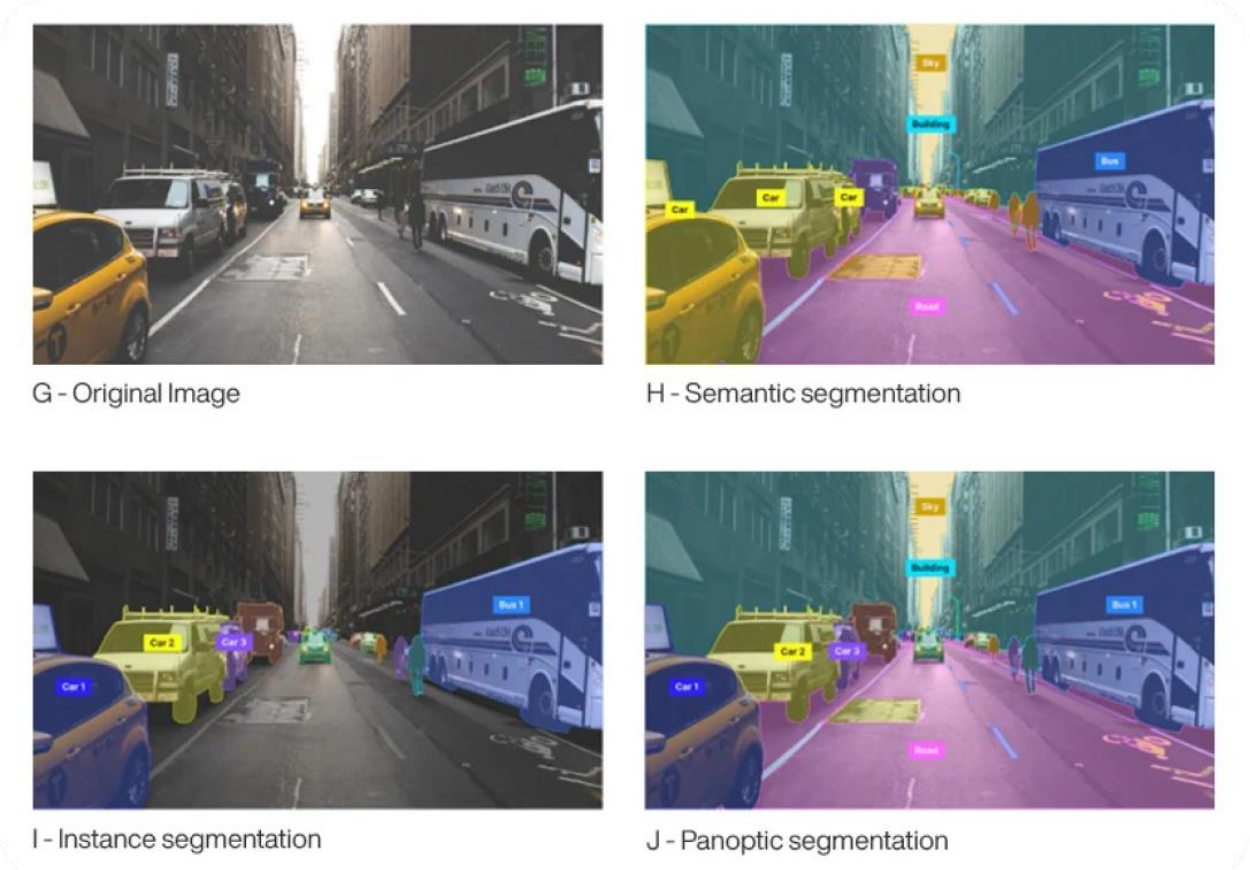

Рисунок 1.3 — Виды сегментации

Первые настоящие успехи в семантической сегментации дал подход FCN (Fully Convolutional Networks), который превратил CNN в «пиксельный классификатор». Он убрал полносвязные слои и научил сеть предсказывать класс каждого пикселя. Это было революцией [9].

Затем пришли U-Net — как хирургические инструменты для биомедицинских снимков, DeepLab — с расширенными свёртками, чтобы не терять детализацию при уменьшении размера карты признаков, и всё более сложные encoder-decoder архитектуры, способные интерпретировать контексты разной глубины [10].

Mask R-CNN же встроил маску в детекцию: сначала определяется объект, затем — его точные границы. Это позволило объединить классификацию, детекцию и сегментацию в один процесс. Паноптические модели, в свою очередь, совмещают две ветви (экземплярную и семантическую) или обучаются на единой функции потерь.

Сегментация — это не про пиксели. Это про смысл, вписанный в каждый пиксель. Машина учится видеть не просто «что есть», а «где это» и «чем оно отличается от остального». В будущем такие модели станут не просто помощниками, а интерпретаторами мира — от медицинских снимков до уличных потоков мегаполисов.

1.2.3 Обнаружение ключевых точек

Обнаружение ключевых точек — это не просто задача локализации. Это акт воссоздания структуры из хаоса — момент, когда алгоритм не просто видит человека, но начинает понимать, где у него плечо, где локоть, как согнута нога и в каком направлении смотрит лицо.

По сути, это как нарисовать каркас, на котором держится динамика тела. Эта задача стоит на границе между компьютерным зрением и биомеханикой, между геометрией и жестом.

Ключевая точка — это ориентир. Точка, которая говорит: «Вот тут угол глаза», «Вот это колено», «Вот сюда опирается тело». Их выбор зависит от контекста: в медицинской визуализации это могут быть суставы, в автомобильной промышленности — края фар и колёс, в биометрии — кончик носа и уголки рта. Но суть одна: найти координаты точек, которые имеют значение.

На выходе алгоритма — набор пар (x,y), указывающих местоположение этих признаков в изображении. И всё бы ничего, если бы не реальный мир — с шумами, перекрытиями, поворотами и недосказанностями.

Оценка позы человека, пожалуй, самый известный и наглядный случай. Требуется, чтобы модель распознала на изображении каждого человека и

«собрала» его из 15–20 ключевых точек: голова, плечи, локти, запястья, бедра, колени, лодыжки. Всё это — как связки на виртуальном скелете.

Набор данных **COCO** задаёт 17 таких точек. Причём задача осложняется: нужно не просто отметить суставы, но и сначала — найти человека. В изображении с множеством фигур модель должна предсказать и ограничивающие рамки, и координаты для каждого из них.

Для данный задачи существуют два основных подхода:

Регрессия координат: нейросеть напрямую предсказывает числа — координаты каждой ключевой точки. Быстро, но чувствительно к масштабу и смещению.

Тепловые карты (heatmaps): вместо координат сеть создаёт карту вероятностей. Максимум — это искомая точка. Такой подход устойчивее: он даёт возможность "увидеть" область интереса, а не точку как число.

Модель **OpenPose** стала первым realtime-решением, способным распознать нескольких людей одновременно — даже если они частично перекрываются. Она строила не только тепловые карты, но и Part Affinity Fields — вектора, показывающие, какие точки должны быть соединены. Так машина училась не только видеть, но и связывать [11].

Более современные архитектуры — HRNet, TokenPose, PoseFormer — добились ещё большей точности. Например, HRNet сохраняет высокое разрешение на всех уровнях, что критически важно для точности локализации.

Область применения ключевых точек поражает своей многослойностью: от развлечений до медицины. В захвате движения система способна регистрировать траекторию тела человека без маркеров — достаточно одного видео, и на экране появляется скелет, точно повторяющий каждый изгиб. В дополненной реальности эти координаты превращаются в живого цифрового двойника: аватары копируют движения в реальном времени, синхронизируясь с пользователем по суставам. В медицине это позволяет анализировать походку, выявлять нарушения осанки, отслеживать прогресс реабилитации без громоздких датчиков — достаточно камеры и модели. В системах распознавания действий те же точки становятся языком жестов: модель определяет, стоит человек, приседает, бежит, или, быть может, машет рукой, подавая сигнал. Там, где раньше требовались годы настройки оборудования — теперь достаточно обучения сети и понимания того, как устроено человеческое тело [12].

На изображениях результаты keypoint detection обычно выглядят как совокупность точек, наложенных на тело, соединённых линиями: получается «палочный человек», но с точностью до миллиметра (рисунок 1.4). Это не просто графика — это структурное представление тела.

Рисунок 1.4 — Пример обнаружения ключевых точек

В контексте компьютерного зрения ключевые точки — это момент, когда пиксели становятся позой, а изображение — предсказанием жеста. Это попытка машины не просто «увидеть» тело, а понять, как оно устроено и что собирается сделать.

1.2.4 Обнаружение объектов

Среди всех задач компьютерного зрения именно обнаружение объектов можно назвать нервным узлом: здесь сходятся и семантика, и локализация, и классификация. Система не просто должна понять, что перед ней, но и где именно это находится, и сколько таких сущностей представлено на изображении. В этом смысле object detection — это не распознавание образа, а полноценная интерпретация сцены.

Представим фотографию улицы: пешеходы, машины, светофоры, фасады зданий, деревья. Модель, способная к детекции, находит каждый значимый объект, обводит его ограничивающей рамкой (bounding box), присваивает метку класса и степень уверенности. Всё — в одном процессе, мгновенно. Это и есть «видение» в машинном смысле: разложить визуальный хаос на управляемые компоненты.

В задаче детекции объектов bounding box (ограничивающая рамка) — это не просто прямоугольник. Это пространственное утверждение: вот здесь находится нечто осмысленное. Вся архитектура object detection, начиная с R-CNN и заканчивая последними версиями YOLO и DETR, строится вокруг этого конструкта.

Типичная bounding box описывается четырьмя параметрами. В классическом axis-aligned формате это координаты центра **(x, y)**, ширина **w** и высота **h**. Альтернативный формат — угловые координаты: x_{min}, y_{min}, x_{max}, ymax — используется, например, для визуализации и расчётов.

Сложные сцены требуют более гибкой геометрии. Здесь появляются Oriented Bounding Boxes (OBB), которые вводят ещё один параметр — угол наклона. Это позволяет описывать наклонные, вытянутые или вращённые объекты (например, корабли на спутниковых снимках или наклонённые знаки в уличных сценах) (рисунок 1.5).

Рисунок 1.5 — Пример bounding box

Bounding box внутри нейросети — это не геометрическая фигура, а результат сложной регрессии. Сначала сеть извлекает признаки, затем предсказывает координаты и класс с помощью головы детектора (detector head). В одноступенчатых сетях (YOLO, RetinaNet) это делается на каждой ячейке feature map. В двухступенчатых (Faster R-CNN) — сначала создаются предложения (RoI), которые уточняются в следующем этапе [13; 14].

Во время обучения bounding box регрессируется через loss-функцию. Это может быть L1/L2 loss, Smooth L1, GIoU loss — они штрафуют за несовпадение предсказания и ground truth.

Bounding box важен не только как геометрия. Это обобщение формы, база для экземплярной сегментации, компромисс между точностью и скоростью. Он прост, быстр и универсален. На его основе строятся маски, и от него зависит вся структура алгоритмической цепочки.

Сегодня видно развитие форматов: anchor boxes, anchor-free, и set-based, в которых больше нет ни якорей, ни NMS, ни эвристик — только внимание и предсказание.

Как было выяснено, в техническом плане объект определяется прямоугольником. На этом контуре строится оценка того, насколько предсказание совпадает с реальностью. Здесь вступает в игру метрика IoU (Intersection over Union) — отношение площади пересечения предсказанной и истинной рамки к их объединённой площади.

Если IoU выше 0.5 и метка правильна — считается, что объект найден. Если ниже — ложное срабатывание. Так появились основные показатели

качества: AP (Average Precision) и mAP (mean Average Precision), где точность усредняется по всем классам и порогам IoU. Современные бенчмарки, такие как COCO, используют строгие пороги: от 0.5 до 0.95 с шагом 0.05, требуя от модели не только угадать класс, но и сделать это точно.

Другая ключевая идея — Non-Maximum Suppression (NMS). В реальности модель склонна выдавать множество рамок для одного и того же объекта. NMS отсеивает лишние, оставляя только наиболее уверенное предсказание и даёт сигнал для работы со следующим объектом.

Object detection — не абстракция. Это основа в системах видеонаблюдения, автономном вождении, робототехнике и даже в поисковых системах изображений. Наборы данных PASCAL VOC, MS COCO и Open Images сыграли роль полигона для этой эволюции. Только COCO включает 80 классов и свыше 100 тысяч изображений, по которым модели учатся видеть — не абстрактно, а предметно.

Object detection сегодня — это end-to-end подходы, где признаки, рамки и классы извлекаются одним механизмом. Вместо ручных решений — обучаемые слои. Вместо алгоритмической цепочки — единая сеть. И пусть классика дала начало этому направлению, именно глубокие нейронные сети превратили компьютерное зрение в силу, способную понимать сцену, а не просто обводить контуры.

1.3 Свёрточные нейронные сети (CNN)

Если бы машина училась видеть с нуля, то первым её уроком, без сомнения, стали бы свёрточные нейронные сети. CNN — не просто архитектура: это настоящая инженерия восприятия, выстроенная на переплетении математической строгости и вдохновений из биологии. Их принцип напоминает о зрительной коре млекопитающих: нейроны активируются лишь в ответ на узкие, простые стимулы — например, на горизонтальную линию, попавшую в поле их рецептивного внимания. Точно так же каждый элемент CNN “наблюдает” лишь часть изображения, но в совокупности они сканируют всё поле зрения.

CNN — это нечто большее, чем просто последовательность матриц. Их суть можно свести к трём фундаментальным механизмам, перекликающимся с концепцией неокогнитрона К. Фукусимы:

Локальные рецептивные поля. Каждый нейрон свёрточного слоя подключён не ко всему входному изображению, а только к его ограниченному фрагменту, например, окну размером 5×5. Это как будто водить лупой по фотографии, выискивая знакомые контуры — нейрон реагирует лишь на ту часть сцены, что в данный момент под его стеклом.

Разделяемые веса. Нейроны, занимающиеся одним признаком, разделяют один и тот же набор весов – фильтр свёртки – для всех позиций на изображении. Один фильтр — это как универсальный шаблон, накладываемый на всё изображение. Он “катится” по кадру, выявляя повторяющийся паттерн — будь то угол, кривая или текстура. Фильтр, обучаясь, находит свой смысл: один реагирует на вертикали, другой — на зигзаги. Именно это свойство делает CNN трансляционно инвариантными: распознанный признак фиксируется, где бы он ни находился в пределах изображения.

Субдискретизация (пулинг). После серии свёрток наступает этап “обобщения”: max-pooling берёт, скажем, квадраты 2×2 и оставляет только пиксель с максимальным откликом. Это как сжимать изображение, сохраняя ключевые штрихи: мелкие детали отбрасываются, но самые важные признаки остаются, да еще и становятся более инвариантными к сдвигам и искажениям. Благодаря пулингу сеть становится устойчивее к небольшим смещениям объекта на изображении.

Опираясь на описанные принципы — локальность восприятия, разделяемость весов и иерархичность представлений — базовая структура свёрточной нейронной сети складывается из повторяющихся блоков: свёрточных слоёв, активационных функций и слоёв субдискретизации. В завершении — один или несколько полносвязных слоёв, действующих как классификатор.

Рассмотрим поэтапно, как обрабатывается изображение внутри CNN — от входа до финального вывода:

Входное изображение. В самом начале сеть получает изображение, например, размером 32×32 пикселя с тремя цветовыми каналами (RGB). Для консистентности размер часто масштабируют до фиксированного. Каждому пикселю соответствуют числовые значения яркости каналов — по сути, массив из трёх чисел. Всё изображение можно представить как трёхмерный тензор: $X \in \mathbb{R}^{\{h \times w \times c\}}$, где h — высота, w — ширина, c — число каналов.

Свёрточный слой. Далее изображение проходит через обучаемые фильтры — маленькие ядра, например, 3×3 или 5×5, которые "скользят" по входу и вычисляют линейные комбинации соседних пикселей. Простейшее аналоговое представление — это как использовать трафарет: накладываем на разные участки и смотрим, где узор совпадёт.

Формально для одномерного сигнала свёртка записывается как:

$$(f * g)(t) = \sum_{\{\tau\}} f(\tau) \cdot g(t - \tau)$$

А для двумерного изображения формула принимает вид:

$$y(i,j) = \sum_{\{u=1\}}^{\{K\}} \sum_{\{v=1\}}^{\{K\}} X(i + u - 1;\ j + v - 1) \cdot W(u,v) + b$$

где X — входной фрагмент, $W(u,v)$ — веса фильтра размером $K \times K$, b — смещение (bias).

Если изображение цветное, фильтр должен охватывать все каналы, то есть глубина фильтра $c = 3$. Каждый фильтр формирует одну карту признаков — тепловую карту активности, где “зажигаются” те области, где обнаружен нужный паттерн. Несколько фильтров формируют целый стек таких карт — это уже следующая глубинная репрезентация изображения.

Каждая карта признаков можно интерпретировать как «отчёт» конкретного фильтра: один «ищет» горизонтали, другой — округлости, третий — текстуры. Совместно они декомпозируют изображение на примитивные геометрические и текстурные особенности.

Функция активации. После свёртки на сцену выходит функция активации. Без неё нейросеть сводится к набору линейных преобразований — сколь угодно сложных, но неспособных аппроксимировать по-настоящему интересные функции.

Наиболее часто используется функция ReLU:

$$ReLU(x) = max(0, x)$$

Её задача — “засечь” наличие признака и отсеять всё неинформативное. Если фильтр сработал — результат положительный и проходит дальше. Если не сработал — результат нулевой, как будто сигнал не обнаружен.

ReLU — как пороговый сенсор: она активирует отклик только тогда, когда сигнал достаточно силён. Это помогает избежать проблем затухающего градиента и ускоряет обучение. График функции представлен на рисунке 1.6.

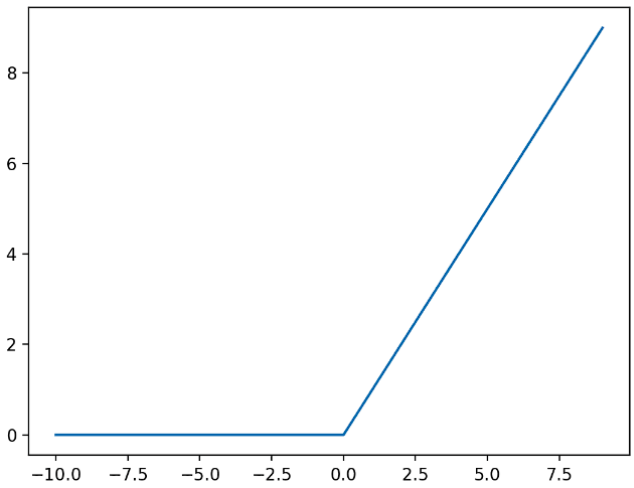

Рисунок 1.6 — График функции активации ReLU

Пулинг. Следующим после свёртки и активации обычно идёт слой субдискретизации, или пулинга. Его задача — не просто уменьшить размерность данных, но и сделать представление признаков более обобщённым, инвариантным к мелким изменениям.

Наиболее распространён — max-pooling, который рассматривает, к примеру, квадраты 2×2 на карте признаков и выбирает из них максимальное значение. То есть, если в блоке значений $\{0, 5, 2, 3\}$, результат будет $\longrightarrow 5$. Таким образом, карта сжимается, но сохраняет наиболее выразительные сигналы, как показано на рисунке 1.7.

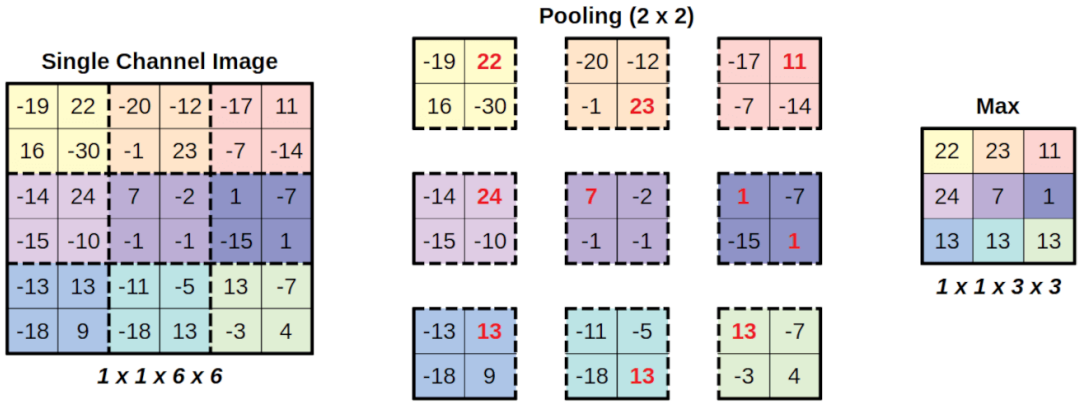

Рисунок 1.7 — Пример работы слоя пулинга

Пулинг существенно сокращает объём данных — размер карты в каждом измерении уменьшается в два раза (если шаг = 2), что снижает вычислительную нагрузку и общее количество обучаемых параметров. При этом сохраняется “суть” изображения, а сеть становится менее чувствительной к небольшим сдвигам и искажениям: даже если объект немного сместился, максимум всё равно его "поймает".

Важно: слой пулинга не содержит обучаемых весов. Данный слой действует как строгий фильтр внимания — выбирает главное, игнорируя второстепенное.

Архитектурная иерархия. Блоки свёртка → активация → пулинг повторяются многократно, формируя глубокую иерархию слоёв. На каждом уровне сеть становится способной распознавать всё более сложные формы:

- **нижние** слои фокусируются на простых элементах — линиях, краях, градиентах;
- **средние** — на более сложных конфигурациях: углы, кривизна, текстура;
- **верхние** — активируются на специфических формах, которые соответствуют деталям объектов: колёса, глаза, антенны.

И, наконец, самые высокоуровневые нейроны срабатывают на цельные структуры — целый автомобиль, кошачью морду, контур лица. Это и есть то, что называется семантическим восприятием: сеть начинает “понимать”, что видит.

Полносвязные слои и финальный вывод. Результат прохождения сквозь слои свёрток и пулинга — множество карт признаков. Например, после последнего свёрточного блока это может быть объём данных размером $6 \times 6 \times 256$. Чтобы использовать их для классификации, они разворачиваются в длинный вектор — своего рода "слепок" изображения. Это называется флэттенинг (flattening).

Этот вектор поступает в полносвязные слои — то есть такие, где каждый нейрон соединён со всеми предыдущими. Это уже классический Multi-Layer Perceptron, который “учится” соотносить конфигурации признаков с метками классов.

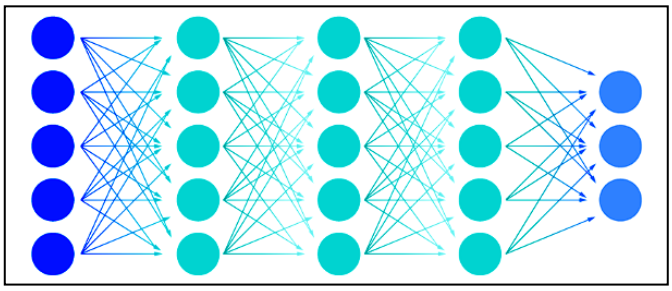

Рисунок 1.8 — Полносвязный многослойный перцептрон

Последний слой обычно содержит столько нейронов, сколько существует классов, и завершается функцией softmax, которая превращает выходы в вероятности:

$$softmax(z_i) = \frac{\exp(z_i)}{\sum_{\{j\}} \exp(z_j)}$$

где z_i — активация i-го нейрона последнего слоя.

Модель возвращает распределение вероятностей по классам, и в качестве предсказания выбирается тот, у которого максимальная вероятность. Это и есть результат — финальный "диагноз", поставленный системой на основе многослойного анализа изображения. Пример работы CNN представлен на рисунке 1.9.

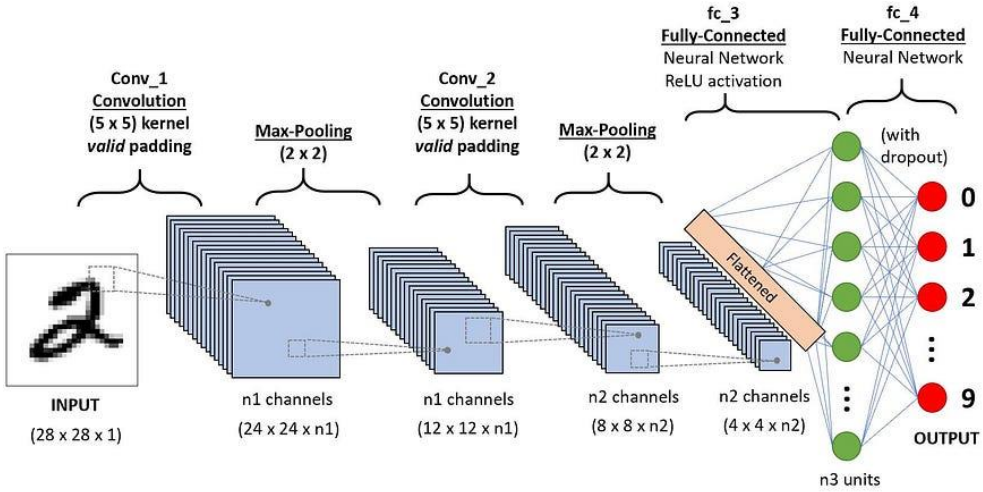

Рисунок 1.9 — Пример работы свёрточной сети

Свёрточные нейросети изначально были заточены под задачу классификации всего изображения. Но вскоре оказалось, что эти же принципы можно переработать для более сложных форм визуального понимания — сегментации и детекции объектов. В этих задачах уже недостаточно одного "ярлыка" на всё изображение: модели требуется выделить где и что именно находится в кадре.

Напомним, в отличие от классификации, где на выходе один класс, сегментация требует полноценной пиксельной карты классов — условную "разукрашку", где каждый участок окрашен в соответствии со своей сущностью.

Чтобы достичь этого, архитектуру CNN превращают в полносвёрточную сеть (Fully Convolutional Network, FCN) [15; 16]. Все полносвязные слои в конце заменяются на свёрточные, и далее используется обратная свёртка (деконволюция) или интерполяция, чтобы восстановить пространственное разрешение (рисунок 1.10).

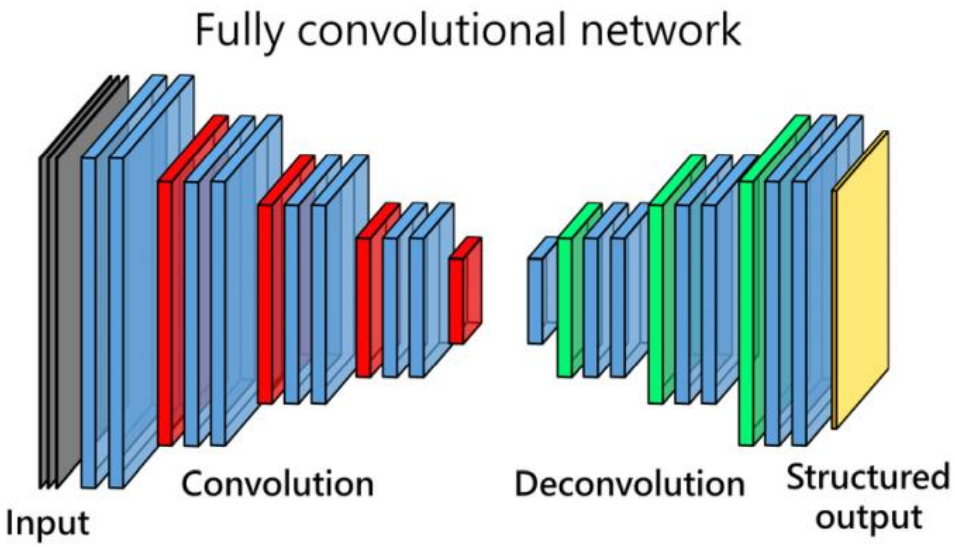

Рисунок 1.10 — Полносвёрточная сеть (FCN)

На выходе получается тензор размером, совпадающим с исходным изображением (ширина × высота), но в каждом пикселе содержится предсказанный класс. Это позволяет получить осмысленную визуализацию.

Архитектурно для этого нередко используют схему **encoder-decoder**:

- **encoder** (обычная CNN) сжимает изображение до абстрактных признаков,
- **decoder** разворачивает эти признаки обратно, восстанавливая карту классов.

Чтобы точнее локализовать границы, применяются skip-соединения — они передают низкоуровневую информацию напрямую в высокоуровневую часть сети, позволяя декодеру не забыть, где именно находились объекты.

В отличие от сегментации, задача детекции объектов требует не только сказать, какие объекты есть на изображении, но и где они расположены.

Именно CNN сделали возможной детекцию в виде сквозной (end-to-end) обучаемой архитектуры. Сначала свёрточные слои извлекают общие признаки из всего изображения, а затем специализированный "head" анализирует эти признаки и предсказывает:

- координаты рамки (например, ⟶ x, y, w, h),
- метку класса,
- вероятность.

1.4. Трансформеры

До некоторого времени казалось, что главенство в задачах компьютерного зрения принадлежит свёрточным нейросетям. Но всего за несколько лет в этом направлении появилось новое направление — **трансформеры**, архитектуры, пришедшие из области обработки

естественного языка. Хотя они были задуманы для работы с текстом, оказалось, что их способность фокусироваться на глобальных зависимостях между элементами делает их удивительно мощными и в анализе изображений. Особенно там, где важно не только локальное, но и дальнее взаимодействие: когда "голова тигра" на одном конце кадра должна узнать "хвост тигра" в другом.

Классическая CNN ограничена локальным рецептивным полем: каждый нейрон "видит" лишь ограниченный фрагмент изображения. Трансформер, напротив, устроен иначе — каждый фрагмент изображения может взаимодействовать с каждым другим. Подобно тому, как человек, глядя на картину, мгновенно улавливает взаимосвязь между объектами в разных её частях, трансформер «запрашивает» у всех других фрагментов контекст, выясняя, что важно и как это связано.

Это реализуется с помощью механизма **самовнимания (self-attention)** — центрального механизма трансформеров, позволяющего вычислять взвешенное взаимодействие между всеми частями входа [17; 18; 19].

Рассмотрим последовательность входных векторов:

$$x_1, x_2, \ldots, x_{\mathrm{n}}$$

Каждый вектор проходит три линейных преобразования, превращаясь в:

- q_{i} — запрос (query),
- k_{i} — ключ (key),
- v_{i} — значение (value).

Это можно выразить так:

$$q_{\mathrm{i}} = W_Q x_{\mathrm{i}}, \quad k_{\mathrm{i}} = W_K x_{\mathrm{i}}, \quad v_{\mathrm{i}} = W_V x_{\mathrm{i}}$$

где W_Q, W_K, W_V — обучаемые матрицы весов.

Далее внимание между двумя элементами i и j вычисляется через скалярное произведение запрос–ключ, масштабированное по размерности:

$$score(q_{\mathrm{i}}, k_{\mathrm{j}}) = \frac{(q_{\mathrm{i}} \cdot k_{\mathrm{j}})}{\sqrt{d}_{\mathrm{k}}}$$

Эти скалярные значения нормализуются с помощью $softmax$ по j, получая веса внимания:

$$\alpha_{\mathrm{ij}} = softmax_j\left(\frac{(q_{\mathrm{i}} \cdot k_{\mathrm{j}})}{\sqrt{d}_{\mathrm{k}}}\right)$$

Итоговый вектор выхода для позиции i — это взвешенная сумма всех значений v_{j}, где веса — это коэффициенты внимания:

$$Attention_{\mathrm{i}} = \sum_{\{j=1\}}^{\{n\}} \alpha_{\mathrm{ij}}\, v_{\mathrm{j}}$$

Матричная форма self-attention для всех позиций одновременно записывается компактно:

$$Attention(Q, K, V) = softmax\left(\frac{QK^{\mathrm{T}}}{\sqrt{d}_{\mathrm{k}}}\right)V$$

где Q, K, V — матрицы, содержащие запросы, ключи и значения для всех входов.

Таким образом, каждый выходной вектор является результатом внимания ко всем другим элементам, и сам решает, на что смотреть.

Представим, как трансформер воспринимает изображение — не как статичную мозаику, а как совокупность потенциальных связей между его фрагментами. Рассмотрим участок изображения, где различим контур лапы животного. Этот фрагмент — не просто пиксели, а носитель "вопроса": "Где остальные части, к которым я принадлежу?"

Каждая другая область изображения формулирует свой "ключ" — описание себя, которое отвечает на этот вопрос. Если где-то в другой части изображения обнаруживается морда, хвост или вторая лапа, система отмечает: "Да, мы из одного мира, из одной формы". В этом случае скалярное произведение между запросом лапы и ключом морды окажется высоким. А значит, соответствующий коэффициент внимания α_{ij} — будет велик.

Дальше механизм самовнимания аккуратно добавит вклад информации из морды в итоговое представление лапы. По сути, фрагмент лапы интегрирует контекст: “рядом — морда, значит, я часть кота”. Такая способность видеть связи без ограничений по расстоянию и без заранее заданных фильтров делает self-attention особенно ценным — в отличие от свёрток, где рецептивное поле строго фиксировано.

Чтобы сделать это восприятие ещё более гибким, трансформеры используют механизм многоголового внимания. Вместо одного набора матриц

Q, K и V, используется несколько — скажем, h независимых "голов", каждая со своей оптикой, со своей перспективой.

Одна из голов может следить за локальными текстурами, другая — за дальними симметриями, третья — за фоновой контрастностью. Все они работают параллельно, проводя разные типы self-attention. По завершении, их выходы конкатенируются (объединяются вдоль признакового измерения) и затем пропускаются через общий линейный слой, возвращая итоговое представление к нужной размерности.

Формально этот процесс можно представить так:

Каждая голова h_i вычисляет

$$Attention_i(Q_i, K_i, V_i) = softmax\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right) V_i$$

Затем все головы объединяются:

$$MultiHead(Q, K, V) = Concat(Attention^1, \dots, Attention_h)W^o,$$

где W^o — обучаемая матрица проекции выходов обратно в общее пространство признаков.

Метафорически это можно представить, как наблюдение за сценой разными глазами: один взгляд замечает силуэт, другой улавливает ритм повторений, третий интуитивно чувствует "что-то важное вдалеке". В итоге получается богатое, слоистое понимание изображения — не просто по шаблону, а с нюансами и ассоциациями. Именно поэтому трансформеры стали революцией не только в языке, но и в зрении. Пример работы трансформера представлен на рисунке 1.11.

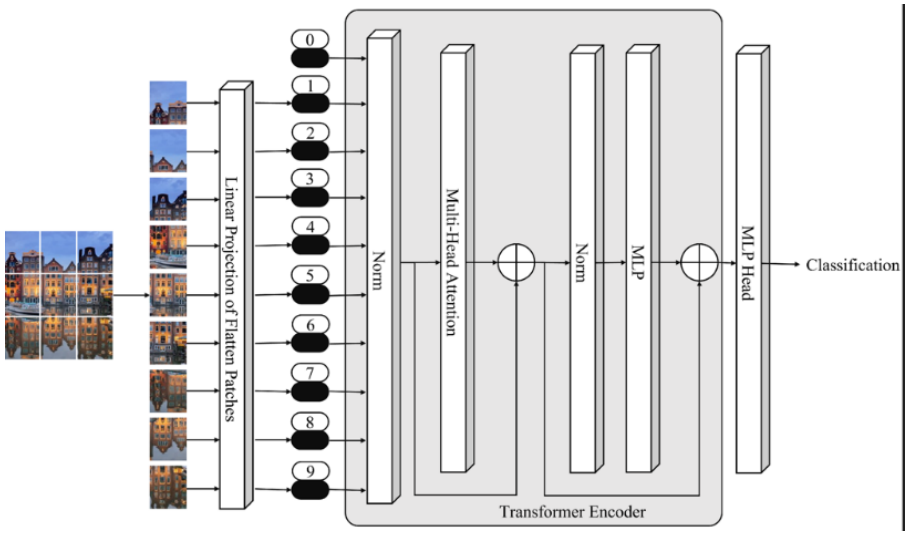

Рисунок 1.11 — Пример работы трансформера

ВЫВОД К ГЛАВЕ 1

В первой главе совершено погружение в фундаментальные строительные блоки компьютерного зрения, начав с постановки самих задач, через которые машина учится видеть: классификация, сегментация, детекция и локализация ключевых точек.

Было рассмотрено, как свёрточные нейронные сети (CNN) стали скелетом этой дисциплины — способные преобразовывать изображения в иерархии признаков, от элементарных контуров до высокоуровневых концептов. На этом фоне рассмотрены задачи сегментации и детекции как естественное продолжение классификации — но с ростом требований к пространственной точности и структурному пониманию сцены.

Также охвачены трансформеры, которые работают с задачами, где требуется более гибкая, глобальная форма внимания. А также их архитектура, вдохновленная лингвистикой, которая позволяет модели "запрашивать" информацию из любых участков изображения, нарушая старые рамки локальности.

В следующих главах будет рассмотрено, как эти архитектуры укладываются в конкретные решения задачи детекции объектов. От эволюции двухступенчатых подходов R-CNN — к одноступенчатым детекторам, где точность и скорость пытаются найти свой хрупкий баланс.

ГЛАВА 2. АНАЛИЗ МЕТОДОВ ДЕТЕКЦИИ ОБЬЕКТОВ ПРИ РЕШЕНИИ ЗАДАЧ КОМПЬЮТЕРНОГО ЗРЕНИЯ

В процессе эволюции методов глубинного обучения в этой области сформировались две концептуально различные парадигмы, два способа взгляда на изображение: двухступенчатые и одноступенчатые детекторы (рисунок 2.1). Эти подходы — не просто алгоритмы, а разные представления о том, как должна работать "машинная интуиция".

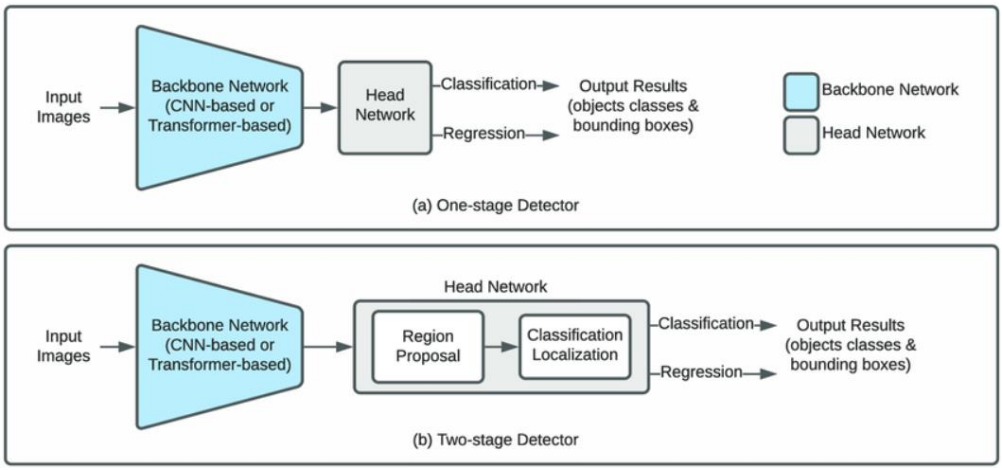

Рисунок 2.1 — Принцип работы одно- и двухступенчатого детектора

Первый путь напоминает работу криминалиста. Сначала система выделяет области, возможно содержащие интересные объекты — своего рода цифровые гипотезы. Это процесс генерации предложений (region proposals), где алгоритм отмечает фрагменты, на которые стоит обратить внимание. Затем, во втором акте, каждая такая область отдельно изучается, классифицируется и уточняется: содержимое, границы, класс объекта.

Такая архитектура, хоть и громоздка, обладает завидной точностью. Данный алгоритм медленно, но методично внимательно рассматривает каждую деталь. Визуально подобный процесс можно представить, как наложение множества прямоугольников — каждый из которых проходит независимую проверку на предмет "что это такое и стоит ли запомнить?"

Второй путь — другой по сути. Он напоминает опытного наблюдателя, который окидывает сцену единым взглядом и тут же говорит: "вот человек", "вот машина", "а это — фонарь". Одноступенчатые детекторы не нуждаются в промежуточной фазе выдвижения кандидатур. Они решают задачу за один проход: локализуют и классифицируют объекты напрямую из исходного изображения.

Этот подход значительно быстрее, особенно на практике — в задачах, где важна реакция в реальном времени (например, беспилотники, видеонаблюдение или AR-приложения). Однако его архитектурная простота

долгое время сопровождалась платой в виде меньшей точности, особенно при наличии перекрытий, сложного фона или маленьких объектов.

На рисунке 2.2 подставлены основные модели этих категорий.

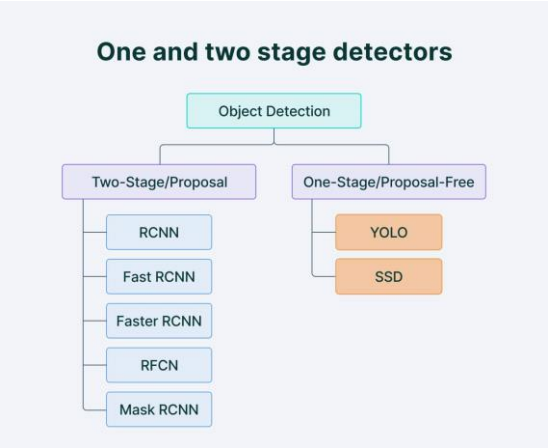

Рисунок 2.2 — Одно- и двухступенчатые детекторы

2.1 Двухступенчатые детекторы

Рассмотрим более подробно логику двухступенчатых (или region-based) детекторов объектов (рисунок 2.3). На первом этапе в работу вступает так называемый механизм генерации предложений (Region Proposal Network или аналог), который просматривает всё изображение и находит фрагменты, которые потенциально могут содержать объекты — их называют областями интереса (RoI). Это похоже на первичную фильтрацию: алгоритм отмечает на изображении, где "что-то может быть".

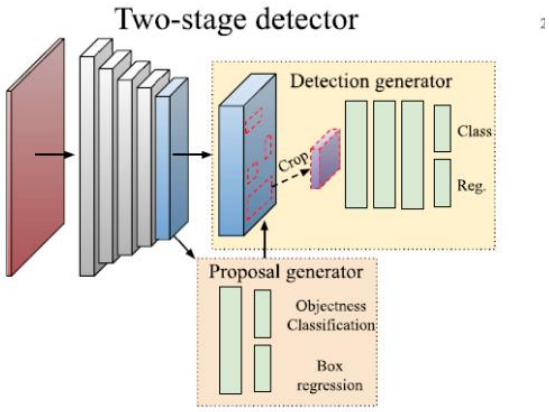

Рисунок 2.3 — Принцип работы двухступенчатых детекторов

Затем каждая такая область проходит через второй этап: классификатор (совмещённый с регрессором координат) уточняет, что именно находится внутри — автомобиль или велосипед, человек или животное — и при необходимости корректирует форму и границы рамки. Таким образом,

детектор концентрируется не на всём изображении сразу, а на наиболее информативных участках.

Такой пошаговый подход оказался революционным в эпоху раннего deep learning. В задачах, где важна не только скорость, но и точность (например, медицинская визуализация или автоматическая разметка спутниковых снимков), двухступенчатые детекторы превосходили по точности любые альтернативы, пусть даже ценой большего времени обработки.

Повторимся, до прихода свёрточных сетей компьютерное зрение опиралось на довольно жёсткую и рутинную схему: скользящее окно, то есть механическое перебирание всех возможных прямоугольников на изображении. Эффективность таких методов была ограничена: точность на PASCAL VOC едва превышала 30–35% mAP, а время на обработку кадра измерялось секундами, если не минутами.

Поворотным моментом стал 2012 год и знаменитая победа AlexNet на ImageNet — сеть, способная сама научиться признакам, превосходящим ручные. Но перенос этой идеи с классификации на детекцию оказался непрост. Причина: локализация. В отличие от классификации, где нужен лишь один глобальный ответ, в детекции требуется одновременно находить, ограничивать и классифицировать множество объектов на одном изображении.

Появившийся в 2014 году **R-CNN** (Region-based Convolutional Neural Network) стал первым по-настоящему действенным мостом между глубокими CNN и задачей локализации [20]. Архитектура, предложенная Girshick и др., разделила обработку на три этапа:

- Генерация ~2000 предложений регионов (с помощью алгоритма Selective Search).
- Масштабирование каждого региона и пропускание его через предобученную CNN (например, AlexNet).
- Классификация и уточнение рамки через SVM + регрессию.

Несмотря на значительный прорыв в точности, R-CNN был крайне медленным: каждая область обрабатывалась отдельно, что приводило к гигантскому времени инференса. Это вдохновило на разработку более быстрых вариантов:

Fast R-CNN (2015): ввёл идею единого пропуска изображения через CNN, после чего применялся слой RoI Pooling — позволяющий извлекать признаковые карты из единой фичи-карты, экономя ресурсы [21].

Faster R-CNN (2016): ещё радикальнее — заменил Selective Search на обучаемую Region Proposal Network, сделав модель полностью end-to-end [22].

Mask R-CNN (2017): добавил к Faster R-CNN отдельную "масочную" ветку, способную производить пиксельные маски, делая возможной экземплярную сегментацию [23].

2.1.1 R-CNN

Схематически R-CNN строится на четвёрке шагов, каждый из которых напоминает ритуал глубокой аналитики:

Генерация регионов интереса. На входе — «сырое» изображение, как нераспаханное поле. Система запускает алгоритм, вроде Selective Search, чтобы сформировать около 2000 прямоугольников, каждый из которых — гипотеза: «А вдруг здесь спрятан объект?». Эти предложения (region proposals) варьируются по размеру и масштабу, стремясь охватить все потенциальные сущности, даже если они отличаются формой и текстурой.

Извлечение признаков. Каждую из предложенных областей растягивают до фиксированного размера (обычно 224×224) и пропускают через предобученную CNN (чаще всего AlexNet). Это всё равно что фотографировать каждое подозрительное пятно через один и тот же фильтр и получать его «фотогенетический» отпечаток — вектор признаков высокой размерности, хранящий семантические паттерны.

Классификация. Далее, каждый регион оценивается — присутствует ли объект, и если да, то какой. В оригинальной R-CNN используется набор SVM, по одному на каждый класс. В отличие от end-to-end моделей, здесь CNN выполняет лишь роль вытяжки признаков, а классификация отделена, как независимая экспертиза.

Точная локализация. Если классификатор подтверждает наличие объекта, запускается линейный регрессор ограничивающей рамки, который корректирует форму и положение рамки, предсказывая смещения:

$$(\Delta x, \Delta y, \Delta w, \Delta h)$$

Кратко, математически, все можно описать так. Пусть изображение представлено как тензор признаков $X \in \mathbb{R}^{\{H \times W \times C\}}$. Region Proposal Network извлекает из него набор прямоугольников $\{r_i\}$, каждый с координатами (x, y, w, h), и вероятностью принадлежности к объекту p_i.

Для каждого региона r_i, извлекается признаковый вектор $f_i = RoIPool(X, r_i)$, после чего предсказывается:

Класс:

$$\hat{y}_i = argmax\, softmax(W_c f_i + b_c)$$

Координаты уточнённой рамки:

$$(\Delta x, \Delta y, \Delta w, \Delta h) = W_r f_{\text{i}} + b_r$$

С последующим преобразованием:

$$x' = x + w \cdot \Delta x$$
$$y' = y + h \cdot \Delta y$$
$$w' = w \cdot e^{\Delta w}$$
$$h' = h \cdot e^{\Delta h}$$

где W_c, W_r — обучаемые веса классификатора и регрессора, соответственно.

Для очистки результата применяется NMS — подавление немаксимальных значений. Суть его — избавиться от повторяющихся рамок, сохранив только ту, которая наиболее уверенно говорит об объекте, использую меру схожести — IoU (Intersection over Union):

$$IoU(B, B') = \frac{Area(B \cap B')}{Area(B \cup B')}$$

То, что раньше считалось хорошим (30–35% mAP), с R-CNN превратилось в базовый уровень. На VOC 2012 модель добилась 53.3% mAP — скачок в +30%. На ILSVRC2013 её результат — 31.4% против 24.3% у OverFeat — окончательно похоронил эпоху скользящих окон.

Метод оказался настолько эффективным, что многие стали воспринимать его как «каноническую» форму глубокой детекции. Но, как это часто бывает в науке — там, где точность торжествует, время замирает. R-CNN был ужасно медленным. Около 2000 областей на изображение, каждая обрабатывается отдельно — в итоге сеть совершает тысячи forward-pass, даже если на картинке только один объект. В среднем — от 30 до 50 секунд на изображение, что делает систему непригодной для реального времени.

Следующая волна моделей взялась за устранение этой неэффективности. Первыми на сцену вышли SPP-net и Fast R-CNN — два разных, но родственных подхода, каждый из которых нацелился на бутылочное горлышко архитектуры R-CNN.

2.1.2 Fast R-CNN

Архитектура Fast R-CNN, была предложенна Россом Гиршиком в 2015 году [21]. Она радикально переосмыслила порядок вычислений и превратила громоздкую систему R-CNN в компактную и совместно обучаемую структуру.

Главный принцип Fast R-CNN заключается в единовременном прогоне изображения через свёрточную нейросеть, что приводит к построению полной карты признаков — тензора размерности: $X \in \mathbb{R}^{\{H \times W \times D\}}$, где D — количество каналов признаков. Эта карта извлекается, например, из последнего свёрточного слоя VGG-16.

Затем, вместо повторной обработки каждого предложения региона, используется специальный слой — RoI Pooling. Он "вырезает" из карты признаков регион, соответствующий координатам предложения, делит его на фиксированную сетку (например, 7×7) и применяет максимальное объединение (max-pooling) в каждой ячейке. Это позволяет привести все регионы к одному размеру, независимо от их исходной геометрии, и передать их в полностью связанные слои.

Ключевое достоинство RoI Pooling в том, что он дифференцируем, а значит, вся архитектура — от свёрточной основы до головы классификации — становится совместно обучаемой, как единая функция.

Fast R-CNN интегрирует два направления предсказаний: вероятностную классификацию по $K + 1$ классам (включая "фон") и регрессию координат ограничивающих прямоугольников для каждого класса.

Вся сеть обучается с использованием единой многозадачной функции потерь:

$$L(p, u, t, v) = L_{\{cls\}}(p, u) + \lambda[u \geq 1]L_{\{loc\}}(t, v)$$

где p — предсказанное распределение вероятностей. А именно:

$$p = softmax(W_c * f + b_c)$$

где u — истинный класс региона, $t = (t_x, t_y, t_w, t_h)$ — предсказанные смещения для ограничивающей рамки, $v = (v_x, v_y, v_w, v_h)$ — истинные смещения, λ — весовой коэффициент, регулирующий вклад локализации, $[u \geq 1]$ — индикатор, исключающий фоновое RoI из потерь регрессии.

Потеря классификации формализуется как кросс-энтропия:

$$L_{\{cls\}(p,u)} = -\log p_u$$

А регрессионная составляющая — как сумма гладких L1-ошибок:

$$\text{smooth}_{\{L1\}(\delta)} = \begin{cases} 0.5\,\delta^2, \text{если } |\delta| < 1, \\ |\delta| - 0.5, \text{иначе} \end{cases}$$

Такой подход оказался значительно более устойчивым к выбросам по сравнению со стандартной L2-регрессией, особенно на этапах раннего обучения, когда разброс координат может быть велик.

Fast R-CNN не просто ускорил R-CNN — он радикально изменил парадигму обучения. Обучение стало единым процессом, не требующим отдельных этапов для настройки CNN, обучения SVM и регрессии. Свёрточные слои теперь адаптировались под задачу локализации, а не только классификации.

Результаты были ошеломляющими. В сравнении с R-CNN, Fast R-CNN:

- был в 213 раз быстрее на тесте (0,3 секунды против 50 секунд на изображение с VGG-16);
- обучался в 9 раз быстрее, за счёт сквозной дифференцируемости;
- показал улучшение точности (по mAP) на PASCAL VOC 2012.

Более того, он превзошёл SPP-net — промежуточное решение — по всем ключевым метрикам, при том, что был проще в реализации и стабильнее в обучении.

Единственным оставшимся “бутылочным горлышком” в Fast R-CNN стал внешний генератор предложений регионов — метод селективного поиска, оставшийся с R-CNN. Хотя сеть ускорилась на два порядка, этот компонент по-прежнему требовал около двух секунд на изображение и не поддавался обучению.

2.1.3 Faster R-CNN

Selective Search, пусть и эффективный, оставался отдельной «чёрной коробкой», находящейся за пределами дифференцируемого контура. Именно эту проблему устранённо решает Faster R-CNN, представленный в 2015 году S. Ren и соавторами [22], благодаря интеграции Region Proposal Network (RPN) — компактного, но мощного свёрточного блока, встроенного в саму модель.

В отличие от избыточной эвристики вроде Selective Search, Region Proposal Network является лёгкой свёрточной подсетью, скользящей по общей карте признаков F, полученной из базовой CNN (например, VGG-16 или ResNet). В каждой позиции скользящего окна на этой карте, RPN одновременно оценивает множество якорей — предопределённых ограничивающих рамок с разными размерами и аспектами. Для каждого якоря она предсказывает:

- вероятность принадлежности к объектному классу;

- параметры регрессии ограничивающей рамки: смещения относительно исходного якоря.

Формально, если в каждой точке карты признаков рассматриваются k якорей, то RPN выводит: карту вероятностей размером $H \times W \times 2k$ и карту регрессий координат размером $H \times W \times 4k$.

Каждый якорь кодируется вектором параметров $t = (t_x, t_y, t_w, t_h)$ которые соответствуют относительным смещениям к координатам якоря (x_a, y_a, w_a, h_a):

$$t_x = \frac{(x - x_a)}{w_a}, \qquad t_y = \frac{(y - y_a)}{h_a}$$
$$t_w = \log\left(\frac{w}{w_a}\right), \quad t_h = \log\left(\frac{h}{h_a}\right)$$

Гладкая $L1$ функция потерь определяется как:

$$smooth_{L1(x)} = \{\, 0.5 * x^2, \text{если } |x| < 1;\ |x| - 0.5, \text{иначе} \,\}$$

Функция потерь RPN записывается как:

$$L_{RPN} = \left(\frac{1}{N_{cls}}\right) * \sum_i L_{cls(p_i, p_i^*)} + \lambda * \left(\frac{1}{N_{reg}}\right) * \sum_i p_i^* * L_{reg(t_i, t_i^*)}$$

После генерации RPN отбирает ~300 лучших предложений (по объектности), которые передаются в модуль Fast R-CNN. Здесь они проходят через RoI Align, классифицируются и уточняются. Ключевая особенность — полное разделение свёрточных вычислений между RPN и Fast R-CNN: обе сети работают с одной и той же картой признаков в тандеме, что делает архитектуру невероятно компактной и эффективной. RPN становится «вторым зрением» модели, подсказывающим, где искать, в то время как Fast R-CNN решает, что найдено.

С появлением Faster R-CNN в 2015 году не только был устранён последний «ручной» элемент в детекционной архитектуре — алгоритм селективного поиска, но и был достигнут качественно новый уровень производительности. При использовании классической на тот момент архитектуры VGG-16 новая модель достигла скорости обработки около 5 кадров в секунду (FPS), включая полный цикл — от извлечения признаков до классификации и регрессии ограничивающих рамок. Это означало более чем пятикратное ускорение по сравнению с Fast R-CNN, и на порядки превышало

показатели оригинального R-CNN, где менее одного кадра в секунду считалось нормой.

С технической точки зрения, производительность Faster R-CNN была обусловлена двумя основными усовершенствованиями:

- Свёрточные признаки извлекались единожды и переиспользовались для всех задач, что обеспечивало высокую эффективность по памяти и времени.
- Сама Region Proposal Network (RPN) была настолько лёгкой и хорошо встроенной, что её вычисления практически не добавляли нагрузки к общей архитектуре.

Однако скорость — лишь часть истории. Точность также сделала шаг вперёд. Faster R-CNN продемонстрировала превосходные результаты на ведущих бенчмарках: PASCAL VOC 2007 / 2012 — средняя точность (mAP) заметно превысила все предыдущие методы; MS COCO — сложнейший на тот момент набор, требующий точной локализации и масштабной обобщаемости, также был покорён.

На соревнованиях ILSVRC и COCO 2015 года модели, построенные на Faster R-CNN, заняли первое место в задачах детекции. Это не было случайным успехом: ключевой причиной стало то, что теперь предложение регионов стало обучаемым, а не фиксированным эвристическим алгоритмом. Модель могла динамически адаптироваться к данным, извлекая закономерности распределения объектов и фокусируя внимание на самых вероятных местах их появления. Такой адаптивности не могли достичь классические нерегулярные методы, основанные на ручных эвристиках.

Таким образом, Faster R-CNN не только закрепила двухступенчатую архитектуру как промышленный стандарт, но и открыла путь к гибридным моделям, где модуль внимания (в лице RPN) и классификатор работают в едином вычислительном контексте. Это стало своего рода переходом от инженерных конвейеров к по-настоящему интеллектуальным системам.

После выхода Faster R-CNN архитектура двухступенчатого детектора стала основой для целого ряда последующих усовершенствований. Исследования 2016–2018 годов концентрировались на усилении всех компонентов: от backbone-сетей до механизмов локализации.

2.2. Одноступенчатые детекторы объектов

Всё начинается с вопроса: а можно ли взглянуть на изображение один раз — и сразу сказать, где и что на нём изображено? Без предварительных догадок, без стадии поиска — просто взять и понять. Именно так рождается идея одноступенчатых детекторов: архитектур, способных производить

обнаружение объектов за один проход по изображению, без явного этапа генерации предложений. Они производят предсказания напрямую, сканируя карту признаков CNN в плотной сетке пространственных координат. Такой детектор — как интеллектуальный фотоаппарат, который не только видит сцену, но и мгновенно аннотирует её смыслами.

В основе одноступенчатых моделей лежит классическая свёрточная нейросеть — аналог позвоночного ствола, от которого отходят «головки» обнаружения (detection head). Эти головки закреплены на разных уровнях глубины и разрешения карты признаков и отвечают за распознавание объектов различных масштабов (рисунок 2.4).

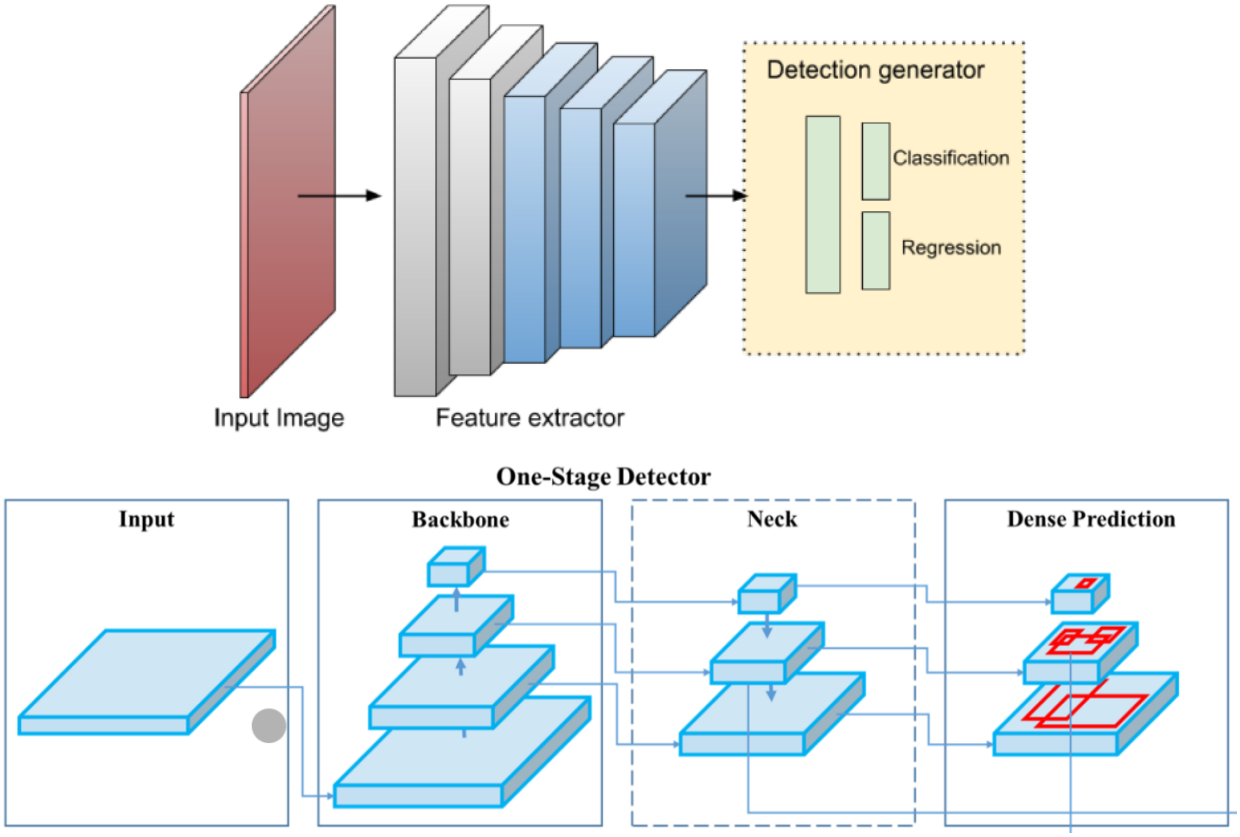

Рисунок 2.4 — Принцип работы одноступенчатых детекторов

Для каждой пространственной точки активации на выходе CNN сеть предсказывает:

- вероятность принадлежности данной позиции к какому-либо из K классов;
- корректировку координат предопределённого «якоря» (anchor box), чтобы точнее подогнать рамку к возможному объекту.

Формально, для каждого якоря r, расположенного на сетке признаков, предсказываются:

Классификационные вероятности: $p_r = (p_r^1, p_r^2, \dots, p_r^K)$.

Регрессионные смещения: $\Delta_r = (\Delta x, \Delta y, \Delta w, \Delta h)$.

Эти смещения преобразуют якорь в уточнённую рамку по следующей формуле:

$$x' = x + w * \Delta x; \qquad y' = y + h * \Delta y;$$
$$w' = w * e^{\Delta w}; \qquad h' = h * e^{\Delta h};$$

где (x, y, w, h) — исходные параметры якоря, а (x', y', w', h') — координаты итогового предсказания.

Идея одноступенчатого обнаружения возникла не на пустом месте. Задолго до того, как R-CNN разрезал задачу на две части, OverFeat уже делал попытки предсказывать рамки и классы напрямую. Однако тогдашние сети были слишком ограничены: точность OverFeat на ImageNet'13 составляла 24.3% mAP, тогда как R-CNN вскоре превысил 31.4%. Прямолинейный подход, хотя и быстро работал, страдал от избытка ложных срабатываний и нехватки локализационной точности.

Однако с ростом глубины CNN, улучшением функций потерь и появлением якорных стратегий стало ясно: одноступенчатые модели — это не только скорость, но и перспектива. Такие модели, как YOLO, SSD и RetinaNet, показали, что можно догнать (а местами и перегнать) двухступенчатые методы по точности, не теряя при этом главного преимущества — скорости.

2.2.1 SSD

В 2016 году Lin и соавторы [14] бросили вызов устоявшейся парадигме двухступенчатого обнаружения. Их модель — SSD (Single Shot MultiBox Detector) — была не просто ещё одним детектором.

Ключевая идея SSD — использовать не один, а несколько слоёв карты признаков для извлечения информации о разных масштабах. Представьте: каждый слой в иерархии свёрточной сети смотрит на изображение под своим 'углом зрения'. Верхние — выхватывают глобальные очертания крупных объектов. Нижние — сосредотачиваются на малых деталях.

На практике SSD с базой VGG-16 использует conv4_3 для мелких объектов, conv7 — для крупных, а затем дополняется кастомными свёртками. На каждом уровне размещаются якоря (anchor boxes) различных форм и размеров. Эти якоря играют роль 'сетей', заранее заброшенных в изображение — в надежде поймать объекты разных пропорций и положений.

В каждом пространственном положении карты признаков надстраиваются поля по умолчанию с разными соотношениями сторон. Например, в ячейке сетки 8×8 может быть четыре таких поля. Для каждого якоря сеть выдает:

1. Вектор вероятностей принадлежности к классам;
2. Вектор смещений, уточняющий координаты.

Это происходит без единого разрыва: одна сеть, один прогон, множество предсказаний. Отсюда:

Классовые вероятности: $\hat{y}_i = softmax(W_c f_i + b_c)$

Регрессионные смещения координат: $(\Delta x, \Delta y, \Delta w, \Delta h) = W_r f_i + b_r$

Преобразование якоря:

$$x' = x + w * \Delta x$$
$$y' = y + h * \Delta y$$
$$w' = w * e^{\Delta w}$$
$$h' = h * e^{\Delta h}$$

Общая функция потерь:

$$L = \left(\frac{1}{N}\right) * (L_{cls} + \alpha * L_{loc})$$

Классификационная потеря:

$$L_{cls} = -\sum_{i \in Pos} \log\left(p_i^u\right)$$

Локализационная потеря (Smooth L1):

$$L_{loc} = \sum_{i \in Pos} \sum_{m \in \{x,y,w,h\}} smooth_{L1}(t_i^m - v_i^m)$$

SSD поражал скоростью. На GPU Titan X модель с входом 300×300 пикселей достигала 58 кадров в секунду, выдавая 72,1% mAP на PASCAL VOC 2007. Увеличение разрешения до 500×500 приносило рост точности до 75,1%, приближая SSD к точности Faster R-CNN — без его задержек.

Но не всё было гладко. SSD, как и другие одноступенчатые модели, сталкивался с крайним дисбалансом: на одно положительное попадание — сотни, а то и тысячи якорей, указывающих на пустоту. Это искажало обучение: сеть слишком быстро училась различать "фон", но терялась, когда дело доходило до редких, малых или перекрывающихся объектов.

Даже с hard negative mining — стратегией, при которой сеть обучается лишь на самых "хитрых" негативных примерах — градиент терял ориентацию.

В результате: снижение полноты и менее уверенная локализация по сравнению с двухступенчатыми конкурентами.

Именно на основе SSD вскоре появится RetinaNet, предложившая элегантное решение — Focal Loss — чтобы сохранить скорость, не теряя точности.

2.2.2 RetinaNET

В 2017 году архитектура RetinaNet поставила точку в одном из самых затянувшихся диспутов компьютерного зрения: может ли однопроходный детектор, обладая лаконичностью, превзойти по точности громоздких двухступенчатых собратьев? Ответ оказался утвердительным и элегантным.

RetinaNet [24] не стремился изобретать всю архитектуру заново — напротив, он унаследовал многие конструктивные элементы от SSD и FPN: иерархию якорей, многомасштабность, ResNet + FPN в качестве базовой сетки. Но именно функция потерь, не архитектура, сделала революцию. Ключевое открытие: радикальный пересмотр того, чему и как должен учиться классификатор в задаче обнаружения.

Классическая перекрестная энтропия слепа к контексту примера — она одинаково штрафует модель за ошибки в простом и сложном случае. Это искажает обучение в задачах с дисбалансом классов: когда большинство якорей не содержат объектов, сеть «привыкает» к фону, переставая замечать редкие, но важные положительные случаи.

Focal Loss вводит регулирующий модификатор в форму стандартной бинарной потери, уменьшая вклад уже верно классифицированных примеров:

$$FL(p) = -\alpha(1 - p)^{\gamma} \log(p)$$

где p — вероятность, предсказанная для истинного класса (например, объект); α — весовой коэффициент, компенсирующий частотный дисбаланс; γ — фокусирующий параметр: чем он выше, тем сильнее снижается вклад лёгких примеров.

Когда $p \to 1$, слагаемое $(1 - p)^{\gamma} \to 0$, тем самым «глуша» градиент. При $p << 1$ вклад в потерю сохраняется. Это позволяет модели игнорировать обыденный фон и сосредоточиться на редких трудностях — истинных объектах в сложном окружении.

Эмпирически установлено: при $\gamma = 2$ и $\alpha = 0.25$ достигается оптимальный баланс.

Простыми словами, RetinaNet — это детектор, который не отвлекается на мусор.

RetinaNet работает как классический одноступенчатый детектор:

1. Использует ResNet-FPN для генерации пирамиды признаков;
2. На каждом уровне размещает априорные якоря — прямоугольники разных масштабов и пропорций;
3. Для каждого якоря предсказывает: вероятность принадлежности к объекту; смещения ограничивающей рамки (4 числа);
4. Все эти предсказания делаются одновременно, за один прямой проход по изображению.

Единая функция потерь объединяет Focal Loss для классификации и Smooth L1 для локализации, применяя её ко всем якорям, соответствующим истине по IoU.

С архитектурой ResNet-101 + FPN RetinaNet достигла: 39.1 AP на COCO test-dev, обогнав Faster R-CNN с той же ResNet на несколько пунктов точности, и при этом обеспечив более высокую скорость вывода.

Таким образом, RetinaNet впервые стерла границу «точность = двухступенчатый подход, скорость = одноступенчатый», а разговоры о неизбежной дилемме «точность против скорости» резко поутихли.

Одноступенчатые методы — до этого считавшиеся пригодными разве что для быстрой фильтрации — заняли центральное место. Появилась новая реальность: если модель обучена правильно, можно получить и скорость, и точность.

Одноступенчатые детекторы продолжили упрощение архитектур, полностью отказавшись от концепции якорей. Вместо предопределённых прямоугольников, такие модели, как CornerNet (2018) и FCOS (2019), предсказывают структуру объекта по-иному — через углы, центры или пиксельные сдвиги. CornerNet определяет bounding box как пару угловых точек, тогда как FCOS регрессирует расстояния от каждого пикселя до сторон блока. Эти подходы устраняют громоздкие якорные параметры (масштабы, соотношения) и приближают задачу к локализации ключевых точек — ближе к сегментации, чем к традиционному box-регрессированию.

Модель FCOS показала: якоря — не обязательны. При сопоставимой точности она сохраняет простоту, изящно обходя сложности сопоставления ground truth с анкерами. В результате детекторы без якорей становятся логичным продолжением философии SSD и RetinaNet — одной сети, минимальных эвристик, полной дифференцируемости.

К середине десятилетия одноступенчатые методы (YOLO, RetinaNet, FCOS) прочно заняли позиции в реальном времени: от автопилотов до смартфонов. Многие из них входят в топ бенчмарков COCO и OpenImages — особенно в задачах, где важны скорость и компактность модели.

ВЫВОДЫ К ГЛАВЕ 2

В этой главе было рассмотрено, как задача обнаружения объектов прошла путь от фрагментарных инженерных решений к целостным, обучаемым архитектурам.

Двухступенчатые детекторы — от R-CNN до Faster R-CNN — предложили принцип: сначала локализовать потенциально значимые области (Region Proposal), затем классифицировать и уточнять. Их сила — в точности. Особенно при обнаружении мелких объектов или сцен с высокой плотностью. Однако ценой этой точности была громоздкость: множественные проходы, внешние модули, раздельное обучение. Faster R-CNN изменил правила, введя Region Proposal Network, и впервые соединил весь процесс в сквозную дифференцируемую схему.

Одноступенчатые детекторы, такие как SSD и RetinaNet, сократили маршрут: они сразу предсказывают классы и координаты объектов, минуя этап предложений. Это открыло путь к real-time, но потребовало решений на математическом уровне: как справиться с лавиной «пустых» якорей? Как отделить значимое от шумового? Ответ дал механизм Focal Loss, концентрируя градиенты на редких и сложных примерах, снижая влияние легко классифицируемых фонов.

В математическом плане каждый шаг в эволюции сопровождался формализацией:

1. Регрессия координат стала стандартной задачей предсказания смещений ($\delta x, \delta y, \delta w, \delta h$),
2. Softmax-потери и их модификации (например, Focal Loss) задали направления для обучения в условиях перекоса классов,
3. Smooth L1 заменила классическую L2, позволив стабильнее оптимизировать локализацию.

Эти элементы превратили архитектуры в системы, способные не просто обнаруживать объекты, а обучаться находить наиболее важное — устойчиво, избирательно и быстро.

В центре этой главы — не борьба архитектур, а демонстрация того, как точка равновесия между скоростью и точностью — это не константа, а подвижная цель, задаваемая контекстом задачи и глубиной математического осмысления.

В следующей главе будет рассмотрена одноступенчатая архитектура — YOLO, которая превратила задачу обнаружения в вопрос одного взгляда, сохранив при этом дух всей эволюции, описанной выше.

ГЛАВА 3. АРХИТЕКТУРА YOLO И ЕЁ РАЗВИТИЕ

История развития YOLO (You Only Look Once) — это не просто техническая хроника сменяющих друг друга моделей. Это метафора того, как идея может пройти путь от вдохновлённой дерзости до зрелой инженерной экосистемы. Начинаясь в 2016 году, как попытка полностью отказаться от региональных предложений и многопроходных схем, YOLO за несколько лет выросла в обширное семейство одностадийных детекторов, сочетающих высокую точность (mAP) с реальной скоростью (FPS), способных работать на любом устройстве — от облачных кластеров до мобильных процессоров.

Каждая версия YOLO — это шаг вперёд не только в цифрах, но и в архитектурной философии. Развитие от YOLOv1 до YOLOv12 (рисунок 3.1) демонстрирует растущую изощрённость моделей при сохранении базового принципа: детектировать всё за один взгляд.

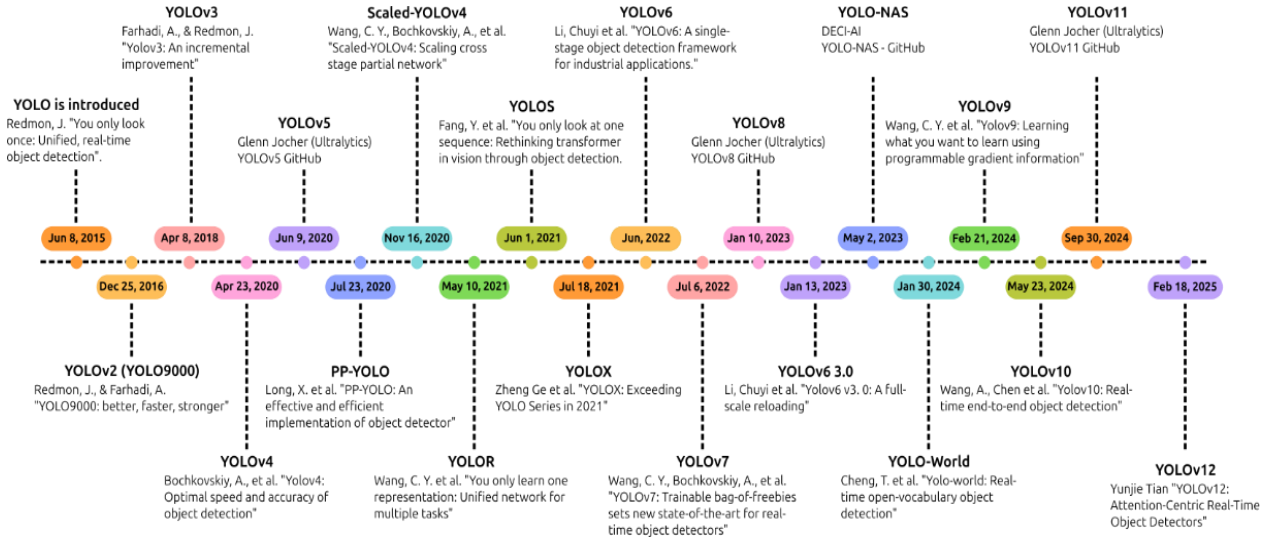

Рисунок 3.1 — История развития моделей семейства YOLO

3.1 Эволюция: от YOLOv1 до YOLOv12

YOLOv1 — взгляд, с которого всё началось. Первая версия YOLO, представленная в 2016 году [25], была своего рода научной провокацией. Вместо того чтобы выделять потенциальные регионы объектов и анализировать их по очереди, как делали RCNN-подходы, YOLOv1 предложила иную философию: видеть и понимать одновременно. Архитектура модели включала 24 свёрточных слоя, за которыми следовали 2 полносвязных, выход которых напрямую предсказывал координаты ограничивающих рамок и вероятности классов (рисунок 3.2).

На техническом уровне YOLOv1 воспринимала изображение как единую сцену, разбивая его на сетку 7×7 ячеек. Каждая ячейка могла

предсказать максимум два объекта. Это придавало системе скорость, которой ранее в детекторах не видели: до 45 кадров в секунду (FPS) на GPU того времени. Но вместе со скоростью пришли и ограничения. Пространственное разрешение выходной сетки не позволило модели корректно локализовать мелкие объекты. Детали терялись, как мелкие ноты в быстрой мелодии.

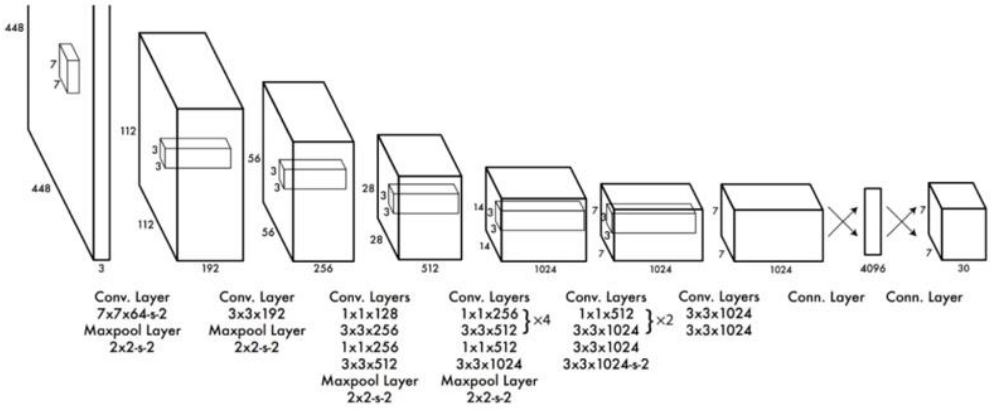

Рисунок 3.2 — Архитектура модели YOLOv1

Несмотря на неточности, модель достигла mAP_{0.5} = 63% на датасете PASCAL VOC2007 — результат чуть хуже, чем у медленных двухэтапных конкурентов, но в десятки раз быстрее. YOLOv1 положила начало революции в объектной детекции, впервые доказав: нейросеть может видеть и действовать моментально.

YOLOv2. Год спустя после оригинального прорыва, в 2017-м, на сцену вышел усовершенствованный преемник — YOLOv2 [26]. YOLOv2, также известная под именем YOLO9000 (в честь числа потенциально различаемых ею классов), стала не просто модернизацией первой версии — она заложила фундамент для будущей универсальности и точности в реальном времени.

Архитектура изменилась радикально. Модель получила в качестве основы более глубокую и компактную нейросеть Darknet-19: 19 свёрточных слоёв с тонкими вставками из свёрточных фильтров размером 1 на 1 пиксель, что снижало число параметров, не жертвуя выразительностью. Но главным поворотом стала интеграция якорей — заранее заданных прямоугольников (anchor boxes) с различными размерами и пропорциями, которые служат шаблонами для поиска объектов. Вместо того чтобы пытаться «угадать» границы объектов с нуля, модель теперь предсказывала смещения относительно этих опорных прямоугольников.

Важным шагом стало внедрение многомасштабного обучения. Во время тренировки размер входных изображений варьировался случайным образом каждые несколько эпох: от 320×320 до 608×608 пикселей. Это придавало

модели гибкость и устойчивость. В тестах, к примеру, на входе 416×416, выходная карта признаков имела разрешение 13×13 — почти в два раза детальнее, чем в YOLOv1. Это обеспечивало более точную локализацию объектов.

Значимым техническим вкладом стал passthrough-слой — архитектурная вставка, которая позволяла задействовать активации из более ранних слоёв (с более высоким пространственным разрешением, например, 26×26), встраивая их в итоговую карту признаков. Это позволило «вернуть» мелкие детали, утраченное при понижении разрешения.

Результат не заставил себя ждать: на том же датасете PASCAL VOC2007 YOLOv2 достигла средней точности почти 78,6% по метрике mAP при пороге IoU 0,5. Это скачок более чем на 15 процентных пунктов по сравнению с первой версией. При этом скорость осталась практически неизменной — от 40 до 60 кадров в секунду в зависимости от разрешения.

Но самым амбициозным элементом YOLOv2 стала версия YOLO9000 — попытка объединить детекцию и классификацию в одной модели. Комбинируя данные из COCO (для детекции) и ImageNet (для классификации), авторы научили модель распознавать более 9000 классов, пусть и с разной степенью точности. Это был первый шаг в сторону универсального визуального понимания, когда модель уже не просто ищет заданные категории, а умеет опознавать гораздо шире.

YOLOv3. Эта итерация стала не просто улучшением — она предложила более гибкий, многоуровневый взгляд на изображение. В основе модели лежал новый бэкбон — Darknet-53, построенный из 53 свёрточных слоёв с остаточными связями, напоминающими ResNet. Эти связи позволили глубокой сети обучаться без деградации градиента, обеспечивая устойчивое распространение информации сквозь десятки слоёв. Все свёрточные блоки были снабжены Batch Normalization и Leaky ReLU, что ускоряло сходимость и повышало стабильность обучения [27].

Но главным новшеством стала многомасштабная предсказательная структура. Вместо одной выходной карты, как у YOLOv1 и v2, здесь было сразу три:

1. Крупная карта 13 на 13 ячеек — для распознавания больших объектов;
2. Средняя карта 26 на 26 — для объектов средних размеров;
3. Мелкая карта 52 на 52 — для мелких объектов, теряющихся на фоне.

Это было достигнуто за счёт передачи признаков через цепь upsample-операций и конкатенаций, то есть сигналы от глубокой, грубой карты признаков “разворачивались” обратно и дополнялись более детализированными. Архитектура модели представлена на рисунке 3.3.

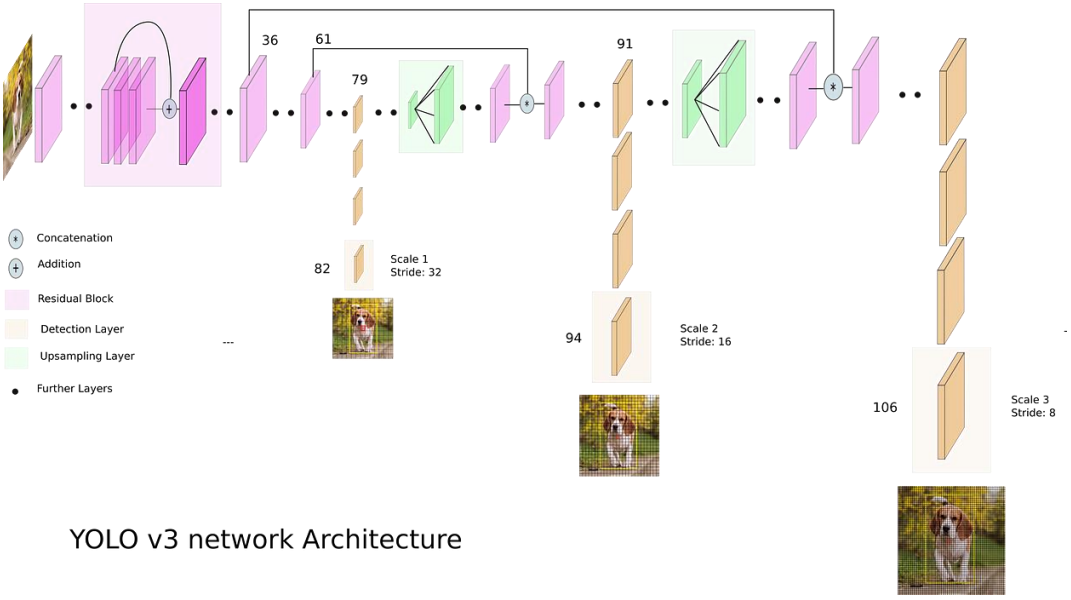

Рисунок 3.3 — Архитектура модели YOLOv3

Модель также изменила подход к предсказанию классов. Вместо softmax-а, назначающего каждому боксу ровно один класс, YOLOv3 использует независимые сигмойды для каждого класса — позволяя предсказывать несколько меток на один объект (если это нужно). Вдобавок каждый якорь теперь снабжается отдельной оценкой объектности — вероятностью того, что рамка действительно содержит объект, независимо от его класса. Эта величина прогнозируется логистической регрессией и затем умножается на вероятности по классам.

Механика фильтрации: как и раньше, перекрывающиеся предсказания проходят процедуру Non-Maximum Suppression — слабые, дублирующие гипотезы исключаются, оставляя наиболее уверенные.

На более строгом и разнообразном датасете MS COCO, где учитываются объекты разных размеров и классов, модель показала около 33–35% mAP по метрике IoU 0.5:0.95 и примерно 55–60% mAP@0.5 — весьма достойные показатели, особенно учитывая, что это достигалось при скорости около 20 кадров в секунду (на изображениях 416×416 пикселей). Вариант YOLOv3-SPP, дополненный модулем пространственного пирамидального объединения перед выходами, обеспечил прирост на несколько пунктов точности за счёт расширения воспринимаемой области признаков.

Важно отметить, что количество параметров в модели возросло — до примерно 62 миллионов — однако благодаря эффективной архитектуре производительность осталась высокой, особенно по сравнению с двухэтапными решениями.

YOLOv4 — прорыв инженерной зрелости. YOLOv4, представленный в 2020 году Алексеем Бочковским [28] и соавторами, стал искусным инженером

— системным, прагматичным, вооружённым всеми доступными средствами. Это не было откровением, скорее — тщательно спроектированный апгрейд, вобравший в себя лучшие архитектурные и обучающие практики своего времени.

YOLOv4 базируется на модифицированном бэкбоне CSPDarknet-53, унаследованном от YOLOv3, но обогащённом механизмом Cross-Stage Partial (CSP) связей (рисунок 3.4). CSP делит поток признаков на две ветви, одна из которых проходит через серию трансформаций, а другая — минует их и объединяется на выходе. Такой подход снижает дублирование вычислений и ускоряет работу, без потери данных.

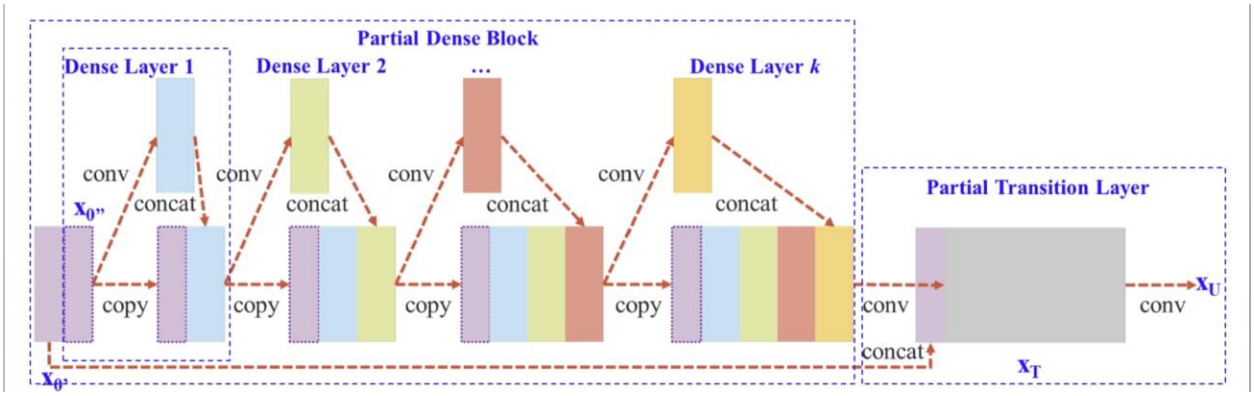

Рисунок 3.4 — Механизм Cross-Stage Partial

В дополнение к этому внедрена новая функция активации Mish, обладающая плавной кривой и улучшающая градиентный поток — в отличие от резкого порога в ReLU. Это делает сеть более чувствительной к сложным паттернам, сохраняя устойчивость при обучении.

«Шея» архитектуры, связывающая бэкбон с выходами, — это модифицированная PANet: она агрегирует признаки с разных уровней и масштабов, позволяя низкоуровневым текстурам и высокоуровневой семантике слиться в единый поток. Вместо сложения, как в оригинале, здесь используется конкатенация — каждый фрагмент информации сохраняет индивидуальность.

Перед финальными выходами встроен модуль пространственного пирамидального объединения (SPP), позволяющий захватывать признаки в рамках разных контекстов (малых и больших полей зрения), а также блок пространственного внимания (SAM), усиливающий значимость областей, в которых сеть “заметила” признаки объектов.

Сами выходы остались якорными: используются anchor boxes на трёх масштабах — крупном, среднем и мелком. Архитектура в целом описывается как: CSPDarknet53 + SPP + PANet + YOLO Head.

Комплексно, но эффективно. Каждая часть дополняет другие: CSP снижает нагрузку, PANet углубляет восприятие, SPP расширяет «обзор», SAM помогает фокусировать внимание, а Mish делает нейросеть «мягче» к сложным сигналам.

YOLOv4 известна не только архитектурой, но и богатым набором приёмов обучения, или, как их называют авторы, «bag-of-freebies» — уловок, улучшающих результат без затрат на инференс. Вот наиболее значимые:

Mosaic-аугментация: четыре случайных изображения объединяются в одно. Это, как если бы сеть “подглядела” сцену, где лошадь прыгает по небу, а человек стоит на крыше — это дезориентирует, но учит быть готовой к чему угодно.

DropBlock: принудительное отключение блоков активаций, аналог dropout, но для свёрточных карт. Это повышает обобщающую способность сети.

Label smoothing: метки классов слегка размываются, чтобы сеть не становилась излишне уверенной в своих предсказаниях.

Cross-miniBatch Normalization (CmBN): вместо того чтобы считать статистику нормализации на одном мини-батче, берутся данные сразу с нескольких — сглаживая шум и повышая устойчивость.

Self-Adversarial Training (SAT): на ранних этапах обучения сама модель искажает входное изображение в тех зонах, где “думает”, что есть объект, провоцируя себя же на более тонкое различие реального от ложного.

Ключевым также стало внедрение новой функции потерь CIoU (Complete IoU), которая при вычислении ошибки ограничивающих рамок учитывает не только перекрытие (как обычный IoU), но и расстояние между центрами прямоугольников и различие в пропорциях. Это делает регрессию более чувствительной к точному позиционированию объектов.

Результаты — новый эталон. На бенчмарке COCO YOLOv4 достигла впечатляющих 43,5% mAP (IoU от 0.5 до 0.95) и порядка 65% AP@0.5, опередив YOLOv3 почти на 8 процентных пунктов. И всё это — при сохранении реального времени: на GPU Tesla V100 модель работала со скоростью примерно 62 кадра в секунду при разрешении 608×608.

Таким образом, YOLOv4 стала де-факто стандартом для практического применения в задачах, где критичен баланс между точностью и скоростью. Она вышла за пределы академических метрик, став используемой в промышленных решениях — от камер видеонаблюдения до бортовых систем дронов.

YOLOv5 — модульность, скорость и массовое принятие. Если YOLOv4 был инженерным триумфом, то YOLOv5 стал воплощением инженерной

зрелости и пользовательского удобства. Появившись в том же 2020 году, почти сразу после релиза YOLOv4, версия от Ultralytics, разработанная G. Jocher [29], не имела за плечами научной публикации, но имела то, что зачастую важнее — открытый исходный код, простой интерфейс и феноменальную адаптацию в индустрии.

YOLOv5 не просто переписали. Её переписали с нуля на PyTorch, отказавшись от C и фреймворка Darknet. Эта смена языка сделала порог входа радикально ниже: любой, кто знаком с PyTorch, мог обучать, тестировать, настраивать YOLOv5.

С архитектурной точки зрения YOLOv5 (рисунок 3.5) унаследовала многое от YOLOv4. Использовался тот же бэкбон CSPDarknet, адаптированный под PyTorch, а также FPN+PANet в качестве “шеи”, соединяющей признаки разных уровней. Встроены и знакомые трюки, такие как mosaic-аугментация, label smoothing, DropBlock, однако самое важное — масштабируемость и модульность.

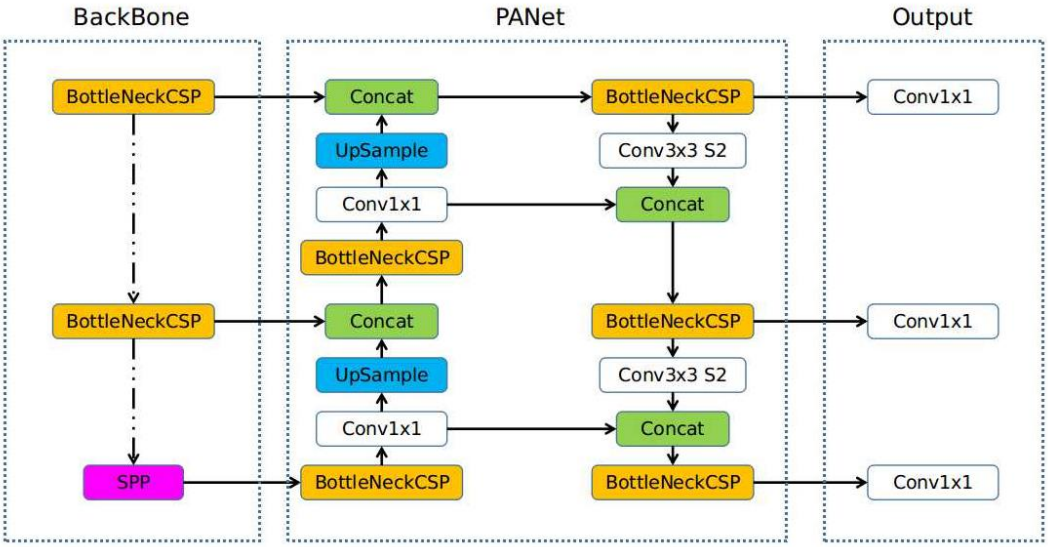

Рисунок 3.5 — Архитектура модели YOLOv5

Одним из факторов массового успеха YOLOv5 стала её интеграция “всё в одном”. CLI-интерфейс, скрипты, визуализация результатов, автоматическая проверка метрик, экспорт весов в ONNX, TensorRT, CoreML, TorchScript — всё это поставлялось “из коробки”, работало без конфигурационного ада. Инженеры могли сконцентрироваться на задаче, а не на дебаге.

Эта версионность означала, что модель подбирается по требованиям проекта, а не наоборот. Если нужно обрабатывать поток с IP-камеры в реальном времени — YOLOv5s. Если накапливаются большие датасеты и нужен максимум качества — YOLOv5x. Эта гибкость особенно проявлялась в задачах, где ресурсы ограничены (Edge AI, IoT, мобильные устройства).

Несмотря на акцент на удобство, YOLOv5 не разочаровала и в цифрах: YOLOv5x, крупнейшая модель, достигала около 50–51% mAP (на COCO при входе 640×640 пикселей), обрабатывая изображение примерно за 0.5 мс на современных GPU — это до 60 FPS.

YOLOv5s, лёгкий вариант, показывал ~37% mAP при скорости более 100 FPS, имея при этом всего около 7 миллионов параметров.

Для индустрии это стало находкой: модель могла запускаться даже на слабом железе, не теряя на точности больше, чем на несколько пунктов. Особенно выделялась скорость обучения: благодаря PyTorch-экосистеме и оптимизированному коду, экспериментальные итерации занимали меньше времени, чем у YOLOv4. Это важный плюс в промышленной разработке, где скорость цикла "обучил — проверил — адаптировал" решает всё.

YOLOv6 и YOLOv7 — два вектора эволюции: индустриальный и академический. К 2022 году развитие семейства YOLO вступило в фазу децентрализации: новые версии начали появляться параллельно — от разных команд, с разными приоритетами. В этой новой главе истории — YOLOv6, ориентированная на промышленную надёжность, и YOLOv7, воплощение научной амбиции. Обе модели стремились к одному — увеличить точность, не жертвуя скоростью, но подходы у них оказались диаметрально противоположными.

Модель YOLOv6, выпущенная китайской корпорацией Meituan [30], была спроектирована как «детектор для реального производства». Здесь не делали ставку на бренд или сообщество — делали ставку на предсказуемость, скорость, воспроизводимость.

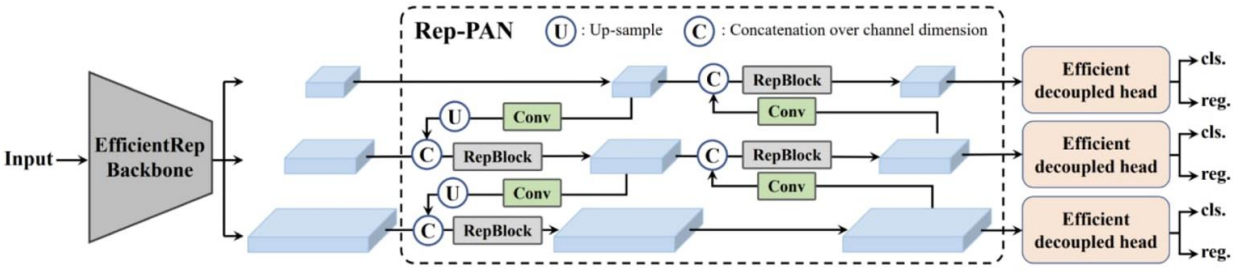

Рисунок 3.6 — Архитектура модели YOLOv6

В основе архитектуры (рисунок 3.6):

EfficientRep Backbone — облегчённый, но выразительный бэкбон, использующий технику репараметризации свёрток: сначала сеть обучается с

разветвлённой структурой, а затем сворачивается в компактный вид для инференса.

Rep-PAN Neck — оптимизированная версия PANet с упором на структурную экономию.

Decoupled Head — вместо общей головы, разделены ветви для классификации и регрессии, чтобы исключить взаимное «помехотворение» градиентов.

Кроме этого:

Anchor-free режим (по центроидам, без фиксированных anchor boxes),

SimOTA — продвинутый алгоритм назначения предсказаний (впервые применён в YOLOX),

SIoU loss — новая метрика качества рамок, учитывающая форму, ориентацию и центр.

YOLOv6 представлена в нескольких версиях (nano, tiny, s, m, l), каждая из которых может работать даже на слабом железе. Так, YOLOv6-s достигала около 43.5% AP на COCO при скорости ~500 FPS (с оптимизацией TensorRT). Модель показала себя особенно хорошо в задачах с ограниченными ресурсами, конкурируя с YOLOv5 и PP-YOLOE.

Если YOLOv6 — это производственный станок, то YOLOv7 — это хронометр, собранный с ювелирной точностью. За ней стоит та же команда, что стояла у истоков YOLOv4, теперь работающая под флагом Института обработки изображений (IFP) Тайваня.

Главное изобретение YOLOv7 [31] — E-ELAN (Extended Efficient Layer Aggregation Network). Это архитектурный паттерн, при котором данные внутри сети циркулируют по множеству «обходных путей». Таким образом, градиенты текут короче, не замирая в глубине. Это позволяет обучать очень глубокие модели без потери обучаемости.

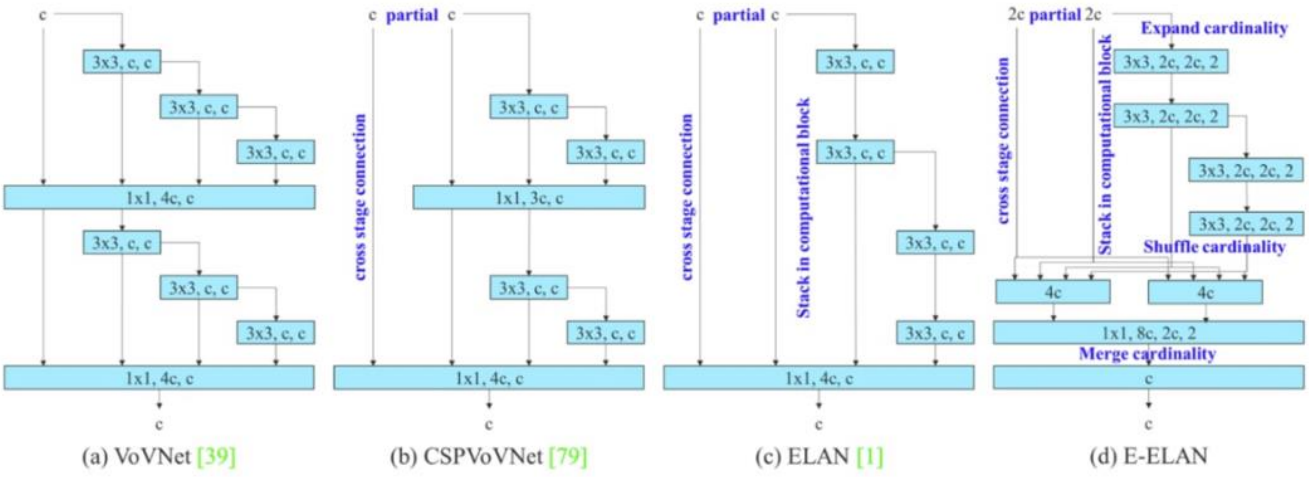

Рисунок 3.7 — Различие архитектур VoVNet, CSPVoVNet, ELAN, E-ELAN

Дополнительно были введены:

- **Gradual Batch Normalization (GBN)** — плавное включение нормализации,
- **Адаптивный масштаб каналов** — показывающий, какие слои можно сделать шире или уже без ущерба точности.

Всё это позволило YOLOv7 при том же (или даже меньшем) числе параметров превзойти своих предшественников. Модель осталась anchor-based, но с возможностью перехода к anchor-free варианту. Вышли облегчённые версии — YOLOv7-tiny и другие. Например, YOLOv7-tiny имела на 39% меньше параметров, чем YOLOv4-tiny, при той же точности. А флагманская YOLOv7-E6E показывала 56–57% AP@0.5:0.95, обгоняя даже более поздние YOLOv8 в ряде конфигураций.

YOLOv8. В начале 2023 года Ultralytics официально представила YOLOv8 [29] — эволюционное продолжение серии, вобравшее в себя всё лучшее от предшественников и доведённое до инженерного блеска. Эта версия окончательно отвязалась от якорных ограничений прежних подходов и вышла на новый виток — не только в архитектуре (рисунок 3.8), но и в философии использования.

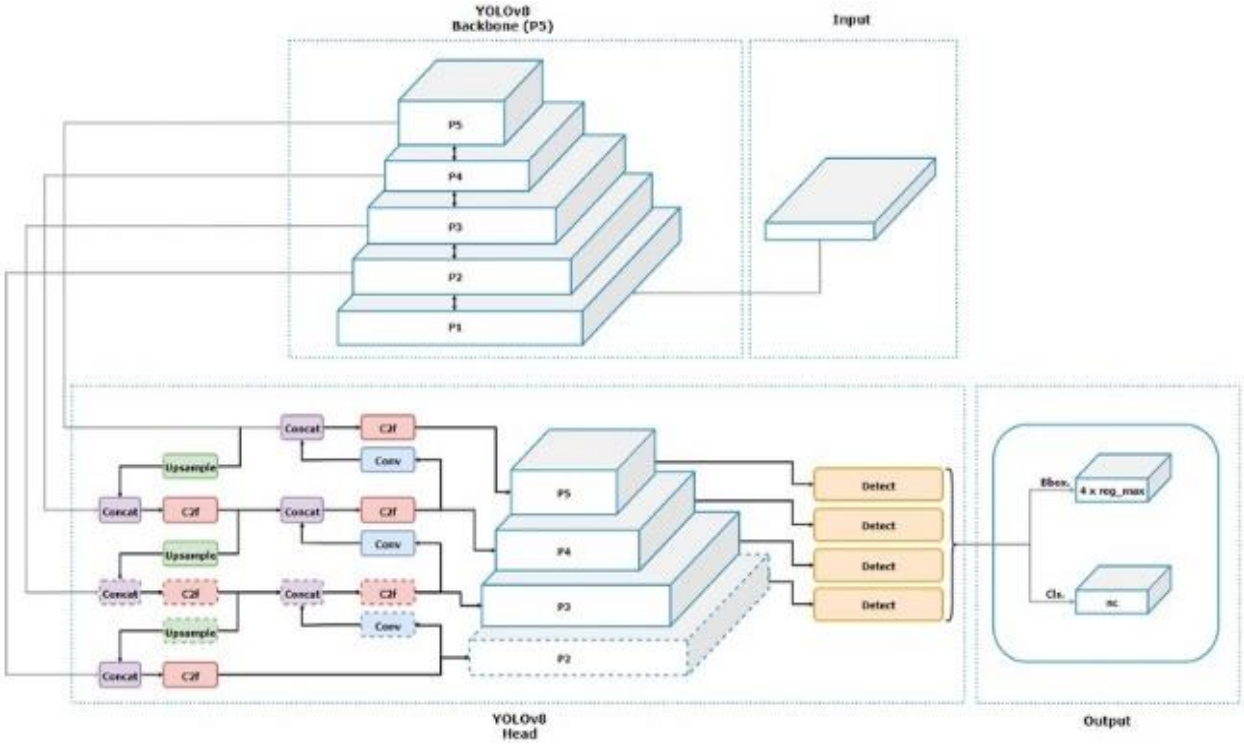

Рисунок 3.8 — Архитектура модели YOLOv8

Главное концептуальное новшество YOLOv8 — anchor-free механизм по умолчанию. Модель больше не зависит от заранее заданных anchor boxes. Вместо этого она напрямую предсказывает координаты объекта — центр,

ширину, высоту — относительно сетки, избавляя разработчика от ручной настройки размеров и пропорций рамок. Это не только уменьшает сложность постобработки и ускоряет инференс, но и повышает обобщающую способность: модель лучше адаптируется к новым датасетам.

YOLOv8 сохраняет принципы CSP-архитектуры, но перепроектирована с упором на компактность и гибкость. Отказ от избыточных связей, обновлённые слои слияния признаков и улучшенная структура головы (decoupled head) — где локализация и классификация разделены на независимые блоки — всё это позволяет добиться лучшего качества детекции с тем же числом параметров.

Особое внимание уделено обратной связи обучения: например, в последние эпохи обучения mosaic-аугментация автоматически отключается, давая модели «сфокусироваться» на чистых изображениях и повысить финальную точность. Это кажется мелочью, но показывает глубину внимания к деталям — как если бы архитектор не только проектировал дом, но и подбирал расстановку мебели в каждой комнате.

YOLOv8 — это уже не просто модель, а целая платформа. В рамках пакета Ultralytics пользователю доступна команда yolo, с которой можно запускать обучение, тестирование, инференс — для задач детекции, сегментации и классификации — в пару строк через конфигурационный YAML.

Модель экспортируется в ONNX, CoreML, TensorRT, легко дообучается и внедряется на edge-устройства. Это делает YOLOv8 особенно привлекательной не только для исследователей, но и для практиков, работающих на мобильных, встроенных и облачных системах.

По метрикам YOLOv8 демонстрирует серьёзное преимущество. YOLOv8x, флагманская версия, достигла примерно 53.9% AP@[0.5:0.95] на COCO (вход 640×640) — примерно на 3% выше, чем предыдущая YOLOv5x при схожей скорости. При этом время инференса ещё сократилось: за счёт упрощённой структуры и отказа от anchor boxes, модель способна выдавать предсказания за доли миллисекунды. А с оптимизацией под TensorRT на GPU NVIDIA A100 заявлена скорость до 280 FPS — показатель, ранее недостижимый для моделей такого размера. Более лёгкие варианты — YOLOv8n, YOLOv8s — демонстрируют впечатляющую эффективность на edge-устройствах и ARM-платформах, обеспечивая реальное время даже на слабом железе.

YOLOv9, разработанная C.-Y. Wang и соавторами [32], представляет собой значительный шаг вперёд в области детекции объектов в реальном времени. Модель внедряет принцип информационного бутылочного

горлышка и обратимые функции, обеспечивая эффективное распространение градиентов и улучшенную сходимость модели. Обратимые функции позволяют сохранять информацию без потерь, что особенно полезно для лёгких моделей, склонных к недопараметризации.

Ключевым новшеством является Программируемая Градиентная Информация (PGI), которая динамически регулирует поток градиентов во время обучения, приоритизируя информативные градиенты для предотвращения потерь признаков. Кроме того, YOLOv9 интегрирует Сеть Лёгкой Архитектуры с Усиленным Градиентом (GELAN) (рисунок 3.9), оптимизируя вычислительные пути для лучшего использования параметров, высокой скорости вывода и точной детекции. С пятью масштабируемыми версиями YOLOv9 обеспечивает баланс между эффективностью и сохранением признаков, делая её высоко адаптируемой для различных реальных приложений.

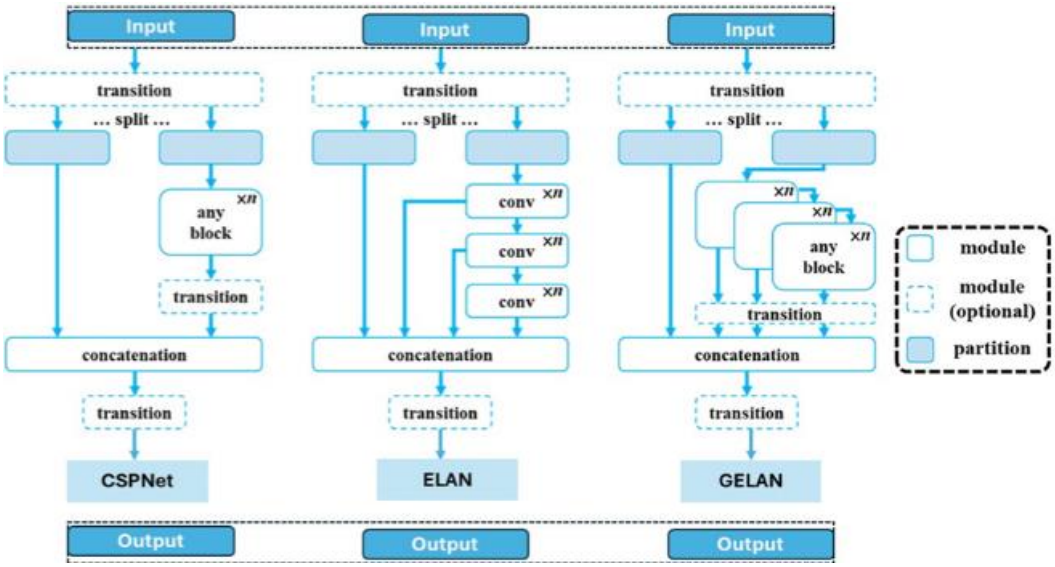

Рисунок 3.9 — Различие между CSPNet, ELAN и GELAN

YOLOv10, разработанная исследователями из Университета Цинхуа [33], представляет собой значительное усовершенствование в архитектуре детекции объектов, направленное на улучшение потока градиентов и вычислительной эффективности. Модель использует улучшенную архитектуру Cross Stage Partial Network (CSPNet) в качестве основы, а также Path Aggregation Network (PAN) в качестве шеи для улучшения слияния признаков на разных масштабах.

Ключевым новшеством является система с двумя головами (рисунок 3.10): One-to-Many Head генерирует несколько предсказаний на объект во время обучения, обеспечивая богатые сигналы для обучения; One-to-One Head используется во время вывода, производя одно лучшее предсказание на

объект, что устраняет необходимость в Non-Maximum Suppression (NMS), снижая задержку и время постобработки.

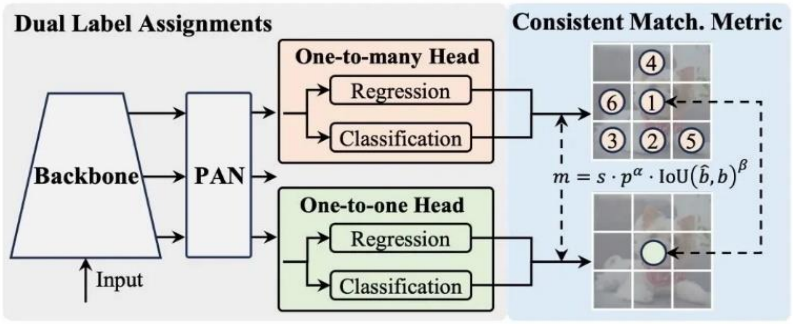

Рисунок 3.10 — Различие в архитектуре модели YOLOv10

Дополнительные улучшения включают лёгкие головы классификации, пространственно-канальное раздельное понижение разрешения, свёртки с большим ядром и модули частичного самовнимания, все из которых улучшают эффективность без значительного увеличения вычислительных затрат. YOLOv10 доступна в пяти масштабируемых версиях, от nano до extra-large, что позволяет применять её в различных сценариях.

YOLOv11, последняя разработка от Ultralytics [34], представляет собой значительный шаг вперёд в области детекции объектов в реальном времени, предлагая непревзойдённую точность и эффективность для различных задач компьютерного зрения. Модель включает улучшенную архитектуру основы и шеи (рисунок 3.11), что повышает возможности извлечения признаков для более точной детекции объектов и выполнения сложных задач.

Ключевые улучшения включают внедрение модуля Cross-Stage Partial с Самовниманием (C2PSA), который объединяет преимущества сетей с частичным перекрёстным этапом и механизмов самовнимания, позволяя модели более эффективно захватывать контекстную информацию на нескольких уровнях. Кроме того, блок C2f был заменён на C3k2, пользовательскую реализацию CSP Bottleneck, использующую две свёртки, в отличие от одной большой свёртки в YOLOv8, что сохраняет точность при повышении эффективности и скорости.

YOLOv11 поддерживает широкий спектр задач, включая детекцию объектов, сегментацию экземпляров, классификацию изображений, оценку позы и детекцию ориентированных объектов (OBB), предлагая пять масштабируемых моделей от nano до extra-large.

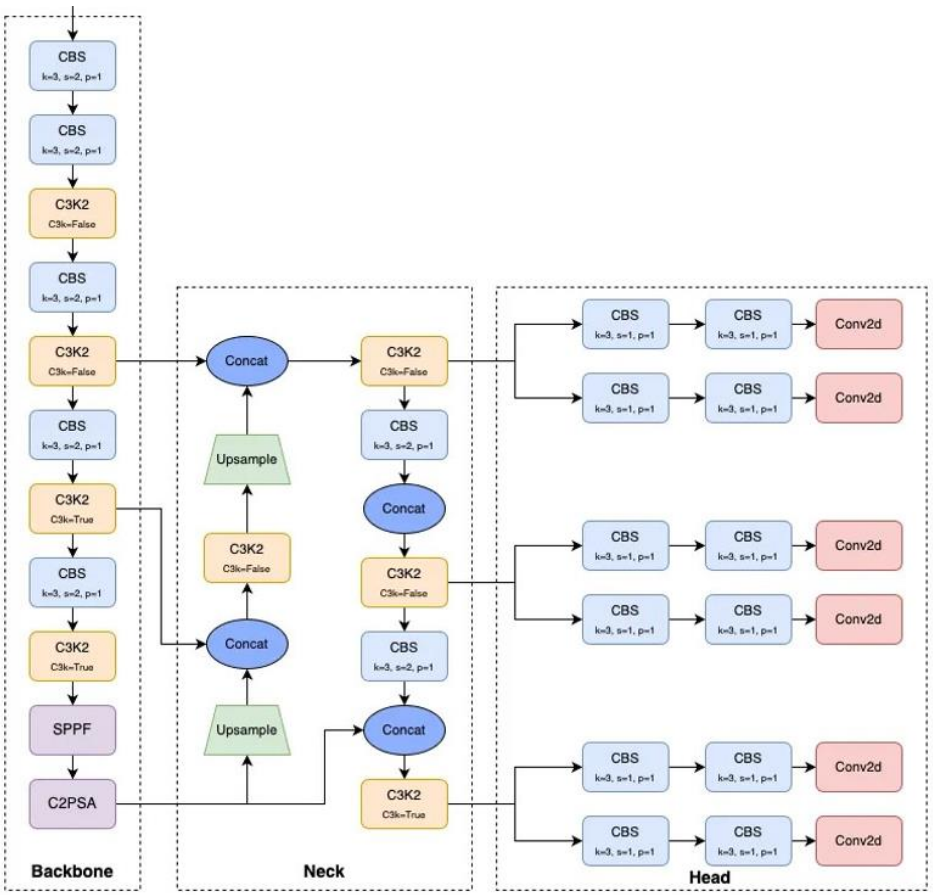

Рисунок 3.11 — Архитектура модели YOLOv11

YOLOv12, последняя эволюция в серии YOLO [35], представляет собой архитектуру (рисунок 3.12), ориентированную на внимание, которая значительно улучшает как скорость, так и точность детекции объектов. Модель продолжает тенденцию предложения пяти масштабируемых моделей (от Nano до Extra Large), что делает её адаптируемой для широкого спектра приложений, таких как детекция объектов, сегментация экземпляров и детекция ориентированных объектов (OBB).

Ключевые нововведения включают модуль Area Attention (A2), который поддерживает большое рецептивное поле при значительном снижении вычислительной сложности, позволяя модели повышать скорость без ущерба для точности. Кроме того, YOLOv12 использует Сети Эффективной Агрегации Слоёв с Остатками (R-ELAN), которые улучшают стабильность обучения и сходимость модели через остаточный дизайн на уровне блоков и оптимизированную агрегацию признаков.

Модель также использует FlashAttention для минимизации накладных расходов на доступ к памяти, сокращая разрыв в скорости с CNN. Она регулирует соотношение MLP с 4 до 1.2, повышая эффективность выполнения, и удаляет позиционные кодировки для более чистой и быстрой модели без потери точности детекции. YOLOv12 также снижает сложность, используя

один блок R-ELAN на последнем этапе основы вместо трёх стеков блоков внимания/CNN. Она заменяет линейные слои на свёрточные слои и нормализацию по батчу, максимизируя вычислительную эффективность и достигая передового баланса между задержкой и точностью.

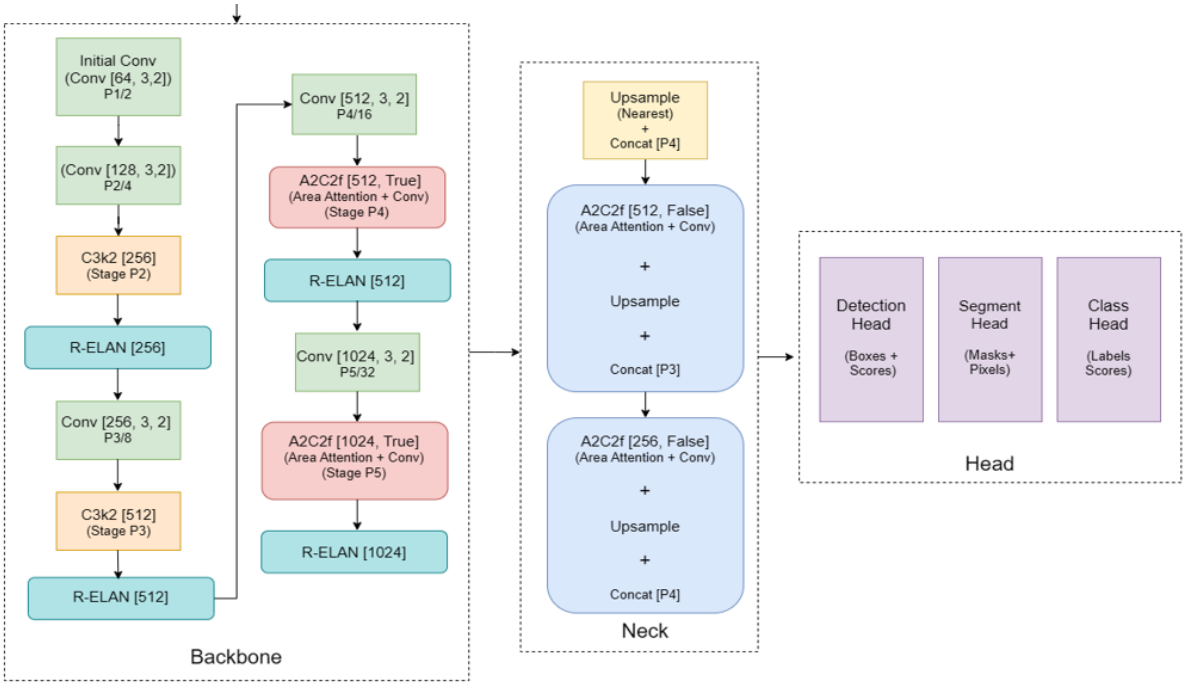

Рисунок 3.12 — Архитектура модели YOLOv12

Versions	Precision	Recall	mAP50	mAP50-95	Preprocess Time	Inference Time	Postprocess Time	Total Time	GFLOPs	Size
YOLOv3u	0.91	0.88	0.943	0.803	**0.5**	6.2	0.4	7.1	**282.2**	**207.86**
YOLOv3u tiny	**0.897**	0.866	0.921	**0.725**	0.7	0.7	0.4	**1.8**	19.1	24.44
YOLOv5un	0.949	0.862	0.948	0.791	1.1	**0.5**	0.4	2	7	5.65
YOLOv5us	0.924	0.882	**0.853**	0.804	1	0.8	0.4	2.2	23.8	18.58
YOLOv5um	0.935	0.887	0.947	0.807	0.6	2.1	0.4	3.1	64	50.54
YOLOv5ul	0.916	0.881	0.948	0.797	0.7	3.3	0.4	4.4	135	106.85
YOLOv5ux	0.932	0.867	0.946	0.797	0.5	6.6	0.4	7.5	246.2	195.2
YOLOv8n	0.932	0.908	0.953	0.794	1.1	0.5	0.4	2	8.2	6.55
YOLOv8s	0.962	0.88	0.943	0.812	0.9	1	0.4	2.3	28.7	22.59
YOLOv8m	0.928	**0.909**	0.953	0.822	0.8	2.5	0.4	3.7	78.9	52.12
YOLOv8l	0.942	0.898	0.957	0.817	0.9	3.9	0.4	5.2	165.1	87.77
YOLOv8x	0.912	0.906	0.953	0.819	0.5	7.1	0.4	8	257.6	136.9
YOLOv9t	0.944	0.875	0.948	0.825	1.3	1.1	0.4	2.8	7.7	**4.93**
YOLOv9s	0.921	0.897	0.956	**0.832**	1	1.2	0.4	2.6	26.9	15.33
YOLOv9m	0.924	0.901	0.952	0.823	0.9	2.8	0.4	4.1	76.5	40.98
YOLOv9c	0.934	0.897	0.96	0.83	0.9	3.4	0.4	4.7	102.7	51.8
YOLOv9e	0.932	0.864	0.944	0.809	0.5	7.6	0.4	8.5	189.3	117.5
YOLOv10n	0.901	0.9	0.936	0.786	1.1	0.7	**0.2**	2	8.2	5.59
YOLOv10s	0.929	0.888	0.947	0.799	0.9	1.1	**0.2**	2.2	24.5	15.9
YOLOv10m	0.91	0.88	0.945	0.8	1	2.4	**0.2**	3.6	63.4	32.1
YOLOv10b	0.905	0.899	0.944	0.809	0.8	3.2	**0.2**	4.2	98	39.7
YOLOv10l	0.922	0.894	0.944	0.8	0.7	3.8	**0.2**	4.7	126.3	50
YOLOv10x	0.96	0.862	0.949	0.819	0.8	6.3	**0.2**	7.3	169.8	61.4
YOLO11n	**0.964**	0.877	0.964	0.802	1.1	0.7	0.4	2.2	**6.3**	5.35
YOLO11s	0.952	0.892	0.959	0.8	1.1	1	0.4	2.5	21.3	18.4
YOLO11m	0.922	0.906	0.96	0.81	0.9	2.4	0.4	3.7	67.7	38.8
YOLO11l	0.937	0.896	**0.965**	0.805	0.9	3	0.4	4.3	86.6	49
YOLO11x	0.908	0.886	0.945	0.795	0.6	6.1	0.4	7.1	194.4	109
YOLOv12n	0.95	0.887	0.96	0.804	0.9	4.8	0.5	6.2	**6.3**	5.34
YOLOv12s	0.948	0.888	0.956	0.809	1.1	5.4	**1.1**	7.6	21.2	18.1
YOLOv12m	0.945	0.875	0.952	0.799	1.1	5	0.8	6.9	67.1	39
YOLOv12l	0.92	0.89	0.952	0.807	1	6	0.6	7.6	88.6	51.2
YOLOv12x	0.935	**0.858**	0.945	0.788	**2.9**	**9.4**	0.5	**12.8**	198.5	114

Рисунок 3.13 — Разница метрик моделей обученных на одной выборке

3.2 Трансферное обучение в YOLO

Одной из ключевых причин повсеместного распространения YOLO стало её редкое в нейросетевом мире качество — пластичность. Эта модель не требует начинать с пустого листа: её можно адаптировать к новой задаче так, опираясь на прошлый багаж знаний. YOLO позволяет использовать предварительно обученные представления для решения узкоспециализированных задач — через механизм трансферного обучения (transfer learning) [36].

В большинстве реальных проектов необходимо распознавание объектов, которых нет в стандартных датасетах вроде COCO. Это могут быть защитные каски, микротрещины, необычные формы животных — всё, что ускользает от универсальных классификаторов. Обучение модели с нуля на таких данных в большинстве случаев избыточно. Гораздо рациональнее взять уже обученную на большом датасете модель и «переучить» её на новом материале.

Допустим, берётся модель YOLOv8s, уже обученная на COCO. Её свёрточные фильтры «знают», как выглядят края, текстуры, формы — универсальные признаки, независимо от типа объектов. Заменяя выходные слои под новые классы и запуская дообучение на собственном, специализированном датасете получаем результат: сеть адаптируется за считанные эпохи, подгоняя распознавание под специфическую задачу.

Системы от Ultralytics предоставляют нативную поддержку transfer learning: достаточно задать путь к предобученным весам и список новых классов. Далее тренировка идёт автоматически, зачастую с заморозкой первых слоёв (freezing) на начальных эпохах — чтобы сохранить универсальные признаки и сфокусироваться на подгонке головы под новые метки. Это снижает риск разрушить уже сформированные фильтры и ускоряет обучение на малых выборках.

Ещё один уровень трансфера — горизонтальный: между моделями разного масштаба. Например, можно использовать веса YOLOv5x для инициализации более лёгкой версии YOLOv5n, сохранив общую «визуальную интуицию» старшей модели. Это ускоряет тренировку на мобильных версиях без потери качества. Такой подход реализован, например, в YOLOv5 и YOLOv8 через механизм Transfer Learning Union [37].

Опыт показывает: даже если в новом домене — например, медицинская визуализация — объекты радикально отличаются от повседневных сцен, многие признаки (края, плотность, симметрия) остаются полезными. Например, YOLOv7, дообученная на рентгеновских снимках, показывает высокую точность при малом числе обучающих изображений, в разы ускоряя сходимость по сравнению с «холодным стартом».

ВЫВОДЫ К ГЛАВЕ 3

С момента появления YOLOv1 в 2016 году парадигма «один взгляд — одно предсказание» перевернула представления о скорости и компактности object detection. Если YOLOv1 была первой моделью, которая научилась «видеть мир мгновенно», то каждое последующее поколение — от YOLOv2 до YOLOv12 — добавляло в эту интуитивную зоркость всё более изощрённую математическую выразительность. Архитектура развивалась от жёстких сеточных ограничений и anchor-боксов до anchor-free и гибких decoupled head, от прямолинейных backbones до изощрённых сетей со вниманием и остаточными связями, от monolithic моделей до масштабируемых конвейеров с transfer learning по умолчанию.

YOLO— это уже не просто модели, а визуальные платформы, которые можно донастроить под почти любую задачу без необходимости реинжиниринга. Возможность переучивать её под новую предметную область — будь то домашние питомцы или трубопроводы — с минимальным количеством данных и вычислений превращает YOLO в универсальный «визуальный орган». При этом инженеру не нужно погружаться в детали архитектуры — transfer learning встроен в саму философию и реализацию (Ultralytics, PyTorch, YAML-конфигурация).

От моделей nano до extra-large, от inference на A100 до edge-устройств — YOLO демонстрирует редкое равновесие между вычислительной компактностью и качеством детекции. Современные реализации (особенно YOLOv8, v9, v10) показывают, что высокоточная архитектура может работать в реальном времени, даже с OBB и задачами сегментации, оставаясь стабильной и переносимой.

С переходом к мультизадачным фреймворкам, таким как Ultralytics, YOLO превратилась в архитектурный стандарт для прикладного компьютерного зрения. Будь то задача классификации, сегментации, обнаружения поворота или позы — YOLO теперь может быть точкой входа в целый стек CV-задач.

ГЛАВА 4. ДАННЫЕ И АННОТАЦИИ

4.1 Анализ данных (EDA), форматы разметки (YOLO, OBB)

Данные — это не просто топливо для модели. Это её мир, её опыт, её вселенная. Каждый датасет в задачах детекции — это не случайный набор изображений, а тщательно отобранная сцена с объектами, аннотированными так, чтобы машина могла научиться видеть. И прежде чем допустить модель к этому визуальному театру, нужно осмотреть площадку: провести разведочный анализ данных — Exploratory Data Analysis, или просто EDA.

EDA — это не формальность, а акт интеллектуального распознавания природы данных. Сколько изображений и каких классов? Каковы размеры объектов? Есть ли перекос — например, тысячи собак, но ни одной лошади? Каковы пропорции рамок, их позиции, плотность и повторяемость? Это всё не абстракция, а конкретные факторы, от которых зависит, будет ли модель отличать велосипед от мотоцикла.

Один из важнейших этапов — визуальная проверка. Не на уровне цифр, а глазами. Взглянуть на разметку: охватывают ли рамки объекты точно? Нет ли сдвигов, пересечений, недоразмеченных сцен? Порой в паре сотен картинок можно найти закономерности, которые иначе бы съели точность модели.

Форматы аннотаций — это язык, на котором датасет общается с моделью. И если язык этот неполный или двусмысленный, результат будет соответствующим.

Наиболее известен и минималистичен формат YOLO. Здесь каждому изображению соответствует .txt-файл с тем же именем. Внутри — строки, каждая из которых описывает объект:

```
1 0.30 0.80 0.10 0.30
2 0.70 0.20 0.30 0.10
```

Первая цифра — номер класса (например, 1 — «автомобиль»), а далее: координаты центра объекта (по ширине и высоте изображения), затем — ширина и высота рамки. Все значения нормализованы от 0 до 1. То есть формат не зависит от разрешения картинки — что делает его особенно удобным при масштабировании.

YOLO — это суть простоты: один объект — одна строка. Один файл — одно изображение. Аннотации компактны, легко читаемы и отлично сочетаются с обучением "на лету". В дополнение к аннотациям идёт YAML-конфигурация (data.yaml), где перечислены пути к изображениям и названия классов.

Однако у YOLO-формата есть историческое ограничение: он создавался для ось-ориентированных рамок (Axis-Aligned Bounding Boxes, AABB). Такие прямоугольники идеально подходят для объектов, стоящих ровно. Но в жизни — и особенно на спутниковых снимках или в съёмках с дронов — всё редко стоит прямо. Лодки наклонены, здания расположены под углом, автомобили повёрнуты. И AABB тут не хватает — рамка захватывает лишнее, накладывается на другие объекты, теряет точность.

В этих случаях на сцену выходит OBB — Oriented Bounding Box. Это прямоугольник, который может быть повернут относительно осей изображения.

Он задаётся либо:

1. координатами центра, шириной, высотой и углом наклона θ;
2. либо (чаще и удобнее) координатами четырёх вершин.

В современных реализациях YOLO — например, в YOLOv8-OBB от Ultralytics — используется второй вариант. Каждая строка аннотации теперь выглядит так:

1 0.30 0.80 0.10 0.30 0.40 0.50 0.70 0.50

Здесь всё по-прежнему нормализовано (от 0 до 1), но вместо четырёх параметров (x, y, w, h) — восемь чисел, задающих последовательные вершины ограничительного четырёхугольника.

Разница между классическим форматом YOLO и его ориентированным аналогом предельно проста, но принципиальна: четыре координаты против восьми. В первом случае — центр прямоугольника и его размеры (ширина и высота), во втором — координаты всех четырёх вершин. Простая геометрия, которая меняет правила игры. Ориентированные рамки способны точно повторить контур объекта, под каким бы углом он ни находился.

Конечно, аннотировать OBB вручную сложнее. Не каждый инструмент разметки позволяет “повернуть” рамку или точно нащупать четыре угла. Однако в ряде задач это необходимость, а не излишество. Возьмём, к примеру, аэрофотосъёмку. С высоты птичьего полёта здания, суда, транспорт — всё это редко выстраивается по осям экрана. Классические рамки охватывают излишний фон, накладываются друг на друга, и модель путается.

Ориентированные рамки снимают этот шум, делая локализацию чище, плотнее, выразительнее. Цена за это — чуть более сложная аннотация, более изощрённые формулы пересечений (уже не простое IoU, а геометрия многоугольников), и необходимость визуализировать результаты как наклонённые прямоугольники.

Современные фреймворки идут навстречу. Один из самых универсальных — MMDetection (от OpenMMLab) — использует формат

COCO JSON как стандарт. В базовом виде он работает с AABB. Но когда речь заходит об ориентированных боксах, в бой вступает плагин MMRotate. Он позволяет использовать датасеты вроде DOTA [38; 39], где объекты аннотированы четырёхугольниками. Внутри MMRotate эта геометрия превращается либо в угол + размеры, либо в массив вершин — и уже на этом обучается модель, предсказывая ориентацию объекта.

Ultralytics также не остались в стороне: в версии YOLOv8 с поддержкой OBB модель способна возвращать координаты углов или угол поворота рамки. Таким образом, даже внутри простой текстовой разметки .txt можно задать геометрию, достаточную для детекции в сложных условиях.

Визуализация ошибок при этом тоже меняется. Вместо привычных наложений прямоугольников теперь требуется отрисовка повернутых контуров, подсчёт пересечений многоугольников, работа с геометрией второго порядка. Это усложнение — но оно даёт модели то, чего ей всегда не хватало: способность “думать под углом”.

4.2 Метрики качества: IoU, mAP, precision, recall

Обучить модель — это только полпути. Вторая, не менее важная задача — понять, насколько хорошо она научилась «видеть» мир: замечать объекты, точно определять их границы, не путать одно с другим и не придумывать того, чего нет. Оценить качество детектора — задача тонкая, требующая особых метрик, выходящих за рамки обычной "точности" (accuracy), привычной в задачах классификации. Здесь нам важны и локализация, и полнота, и аккуратность. Поэтому в арсенале компьютерного зрения — несколько взаимосвязанных, но разных метрик [40; 41; 42].

IoU: мера «согласия». В основе всего лежит IoU (Intersection over Union) — метрика, которая измеряет степень пересечения предсказанной рамки с истинной. Это своего рода шкала доверия: насколько хорошо модель угадала не только что это за объект, но и где он находится.

Формально:

$$IoU = \frac{\text{Площадь пересечения рамок}}{\text{Площадь их объединения}}$$

Если рамки идеально совпали — IoU = 1. Если не пересеклись вовсе — 0. Значение 0.5 означает, что общая зона перекрытия — ровно половина объединённой площади.

Это можно представить визуально: истинная рамка — как цель, предсказанная — как прицел. Чем плотнее они совпадают, тем ближе IoU к единице (рисунок 4.1). В реальной задаче редко удаётся достичь идеала, но

чем выше IoU — тем выше уверенность в том, что модель правильно нашла объект и правильно обвела его.

На практике задаётся порог IoU, начиная с которого предсказанная рамка считается «успешной». Принято считать True Positive (TP) те боксы, у которых IoU с реальным объектом выше, скажем, 0.5. Ниже — False Positive (FP). Если объект есть, а рамки нет — это False Negative (FN).

И вот тут начинаются тонкости. Повысим порог до 0.75 — и многие рамки, которые ранее считались точными, теперь станут ошибками (FN). Уменьшим до 0.3 — и получим много ложных совпадений (FP). Порог — это компромисс между строгостью и всеохватностью.

Более того, если модель выдала несколько рамок на один объект, только самая точная (с максимальным IoU) засчитывается как TP. Остальные становятся FP. Это делается, чтобы модель не «накручивала» себе очки за счёт множества близких рамок.

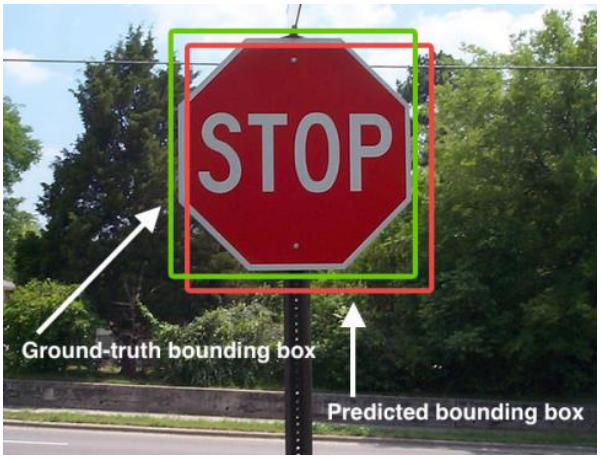

Рисунок 4.1 — Пример различий между идеальной BB и предсказанным

В реальной практике для анализа используют визуальную раскраску предсказаний:

1. TP — зелёная рамка (успешное попадание);
2. FP — красная (ошибочное срабатывание);
3. FN — жёлтая пунктирная (объект пропущен).

Эти цвета наглядно показывают, где модель точна, где ошибается, а где — слепа. Подобный анализ помогает не только оценить качество, но и понять причины ошибок.

Precision и Recall: точность и полнота. На этой основе рассчитываются две фундаментальные метрики:

Precision (точность) — доля предсказаний, которые оказались верными:

$$Precision = \frac{TP}{(TP + FP)}$$

Recall (полнота) — доля истинных объектов, которые были найдены:

$$Recall = \frac{TP}{(TP + FN)}$$

Высокая precision означает, что модель почти не ошибается. Высокий recall — что она почти ничего не пропускает. Но достичь одновременно максимума обеих метрик трудно: как в любой системе сигналов, всегда есть обмен между “не прозевать” и “не переусердствовать”.

Так как precision и recall нередко находятся в конфликте — выигрыш одного влечёт за собой потери другого — вводят **F1-score**, чтобы измерить баланс. Это гармоническое среднее, чувствительное к перекосам:

$$F1 = 2 \times \frac{(Precision \times Recall)}{(Precision + Recall)}$$

Если хотя бы одна из метрик близка к нулю, F1 сразу падает. Это делает его отличным индикатором дисбаланса в поведении модели, особенно на уровне настройки порогов. Например, если модель слишком легко “верит” в свои предсказания, она может радостно выдавать сотни боксов (высокий recall), но среди них будет много ошибочных (низкий precision). И наоборот — слишком строгая модель будет молчать там, где стоило бы сказать хоть что-то.

AP и mAP: интегральная оценка. Вместо того чтобы оценивать модель на одном пороге, принято измерять качество по всей кривой precision-recall. Для этого строится график, где по оси X — recall, по оси Y — precision, а затем вычисляется площадь под кривой — это AP (Average Precision).

В задачах общего назначения (например, COCO) применяют mAP (mean Average Precision) — среднее значение AP по всем классам. Причём не только на одном IoU-пороге (например, 0.5), а по диапазону от 0.5 до 0.95 с шагом 0.05. Так получают строгую, но честную картину.

$$mAP@[0.5{:}\,0.95] = \text{ср. зн. } AP \text{ на порогах } IoU = 0.5, 0.55, \ldots, 0.95$$

Такой подход делает модель более универсальной. Она не просто хорошо локализует “на глаз”, но умеет быть точной до пикселя.

Чтобы это прояснить, представим модель, которая почти всегда “попадает” в нужную область, но не точно (скажем, рамка немного сдвинута).

У неё может быть:

- mAP@0.5 = 0.8 — почти все объекты хоть как-то найдены;
- mAP@[.5:.95] = 0.4 — а вот локализация небрежна.

Другой пример — точная, но избирательная модель:

- mAP@0.5 = 0.6

- mAP@[.5:.95] = 0.55

Это значит, что она редко ошибается, но не охватывает всё. Подобные различия имеют прикладной смысл: для видеонаблюдения важнее recall, а для роботизированной манипуляции — точность позиционирования (высокий IoU).

Для оценки работы моделей, визуальный анализ — ключ. Исследователь может построить:

- PR-кривые по каждому классу — понять, где точность резко падает;
- confusion matrix — увидеть, какие классы путаются;
- визуализацию TP / FP / FN — зелёные, красные, жёлтые рамки.

Кроме того, анализ средних значений IoU по предсказаниям даёт понять: модель плохо локализует (низкий IoU) или модель просто пропускает (низкий recall).

ВЫВОДЫ К ГЛАВЕ 4

Детектор видит не изображение, а то, как оно описано. В этой главе стало ясно: успех модели начинается не с архитектуры, а с разметки. Даже самая совершенная сеть теряет силу, если «глаза» у неё затуманены — аннотациями с ошибками, перекошенными классами или бедной геометрией рамок.

Разведочный анализ (EDA) — это не скучная статистика, а первое знакомство с датасетом: особенности, перекосы, шаблоны. Попытка найти не просто числа, а намёки на будущие ошибки модели.

Форматы разметки — это языки, на которых модель обучается видеть. YOLO — прямолинейный, лаконичный, но ограничен осевыми рамками. OBB — более выразительный, гибкий, особенно для наклонных объектов, как в аэросъёмке, что даёт точность, которую AABB не обеспечит.

Метрики, такие как IoU, precision, recall и mAP, не просто измеряют качество — они вскрывают поведенческую стратегию модели. Одни детекторы не замечают мелкие объекты и молчат. Другие — слишком разговорчивы. Только тонкий баланс между полнотой и точностью даёт по-настоящему надёжный результат. Здесь каждый процент mAP — показатель умения модели видеть в сложных сценах и не путаться в деталях.

Итог: точная аннотация и вдумчивая оценка качества — не подготовительные этапы, а фундамент всей системы. Именно с них начинается зрение нейросети. И если помочь смотреть правильно — она будет видеть лучше ожидаемого.

ГЛАВА 5. Практическая реализация и анализ результатов

5.1 Используемые фреймворки и окружение

Архитектура эксперимента опиралась на два ключевых программных компонента: библиотеку Ultralytics и инструменты из экосистемы scikit-learn [43]. Первая — это современный, гибкий фреймворк, разработанный вокруг моделей YOLO [44], включая последние версии с поддержкой ориентированных ограничительных рамок (OBB). Благодаря унифицированному API и встроенной поддержке аннотаций, визуализаций и экспорта моделей, Ultralytics стал ядром реализации.

Вторая составляющая — sklearn — использовалась для дополнительного анализа метрик, построения диаграмм, расчёта корреляций между показателями и генерации сравнительных таблиц. Визуализация ошибок, подсчёт precision/recall, F1 и построение confusion matrix были реализованы с её помощью.

Эксперименты проводились на локальной машине с поддержкой CUDA и параллельно в Colab и Kaggle, где всё окружение и зависимости автоматически инициализировались при помощи встроенных скриптов. Все результаты сохранялись в структурированных каталогах (results/, onnx_models/), обеспечивая удобную репликацию и анализ.

Для систематизации экспериментов и наглядного представления всех результатов использовалась платформа Weights & Biases (W&B). Через автоматическую интеграцию с библиотекой Ultralytics логировались основные метрики, графики обучения, визуализации предсказаний и сравнения между архитектурами. Это позволило выявлять закономерности, отлавливать неустойчивое поведение моделей и быстро переключаться между сессиями даже при сбоях в обучении. W&B выступила в роли аналитической среды и системы восстановления при повторных запусках.

5.2 Датасеты: структура, цель и особенности.

PSB: PCB Segment Benchmark (электронные компоненты). Датасет PSB [45] включает 190 изображений, фиксированного разрешения 1024x1024 пикселя. Он предназначен для задач обнаружения и классификации электронных компонентов на микросхемах. Разметка выполнена в формате DOTA с возможность совмещения с MMRotate, что требует дополнительной работы с переводом к стандарту OBB, когда каждая аннотация содержит класс и координаты ограничительной рамки. Всего присутствует 28 классов, таких как CE, J, C, IC-SOP, RS и др.

Классическое распределение: от единичных вкраплений (например, IC-SON, 2 экземпляра) до тысяч примеров в категориях J и C. Типичный кадр — лабораторная микросъёмка с минимумом шума, высокой чёткостью и ясным разделением объектов. Боксы практически не пересекаются, сцены строго контролируемы. Источник данных — Kaggle, лицензия MIT. Применение: автоматическая диагностика плат, обнаружение бракованных модулей, визуальный мониторинг производства.

ShipOBB: детекция судов в аэрофотоснимках. ShipOBB [46] содержит 874 изображения с нефиксированным разрешением. Объекты представлены двумя классами: smallShip и bigShip. Типичный кадр — съёмка с беспилотников: открытые морские пространства, прибрежные зоны, порты. Применение датасета — обучение моделей детекции для задач морского мониторинга, в частности для систем беспилотных воздушных судов.

Всего размечено 877 объектов: 486 малых судов и 391 крупное. Формат OBB позволяет точно описывать угловую ориентацию кораблей, что критично при поворотах или частичной видимости. Данные аккуратно размечены, пересечений рамок почти нет. Лицензия CC0 (Public Domain).

DentalXRAY: рентгенограмма зубов и заболеваний. DentalXRAY [47] — это медицинский датасет, включающий 644 изображения рентгеновских панорам челюсти детей. Разрешение варьируется от 1500x1000 до 2000x1000 пикселей. Каждое изображение сопровождается аннотациями OBB для описания различных стоматологических аномалий.

Классы представляют собой диагнозы: от Decayed Tooth и Root Canal Treatment до Enamel Hypoplasia и Abnormal Eruption of Teeth. Баланс классов умеренно смещён: от нескольких экземпляров (например, Congenitally Missing Teeth) до сотен (Caries Restoration, Decayed Tooth). Типичный снимок — синий или тёмно-серый фон с белыми контурами зубов, минимальный шум, но высокое разнообразие случаев.

Данные взяты с Kaggle, лицензия CC BY-NC 4.0. Применение — обучение моделей поддержки диагностики в клинической практике, разработка умных стоматологических устройств.

5.3 Сравнительный анализ результатов моделей

Идея всего проекта — обучить модели YOLOv8-obb, YOLOv11-obb, YOLOv12-obb всех возможных конфигураций на трёх разных датасетах и провести сравнительный анализ полученных результатов. Но, несмотря на формальную симметрию условий, в ходе практического обучения выявился ряд существенных различий в поведении моделей разных семейств и размеров:

Модель YOLOv12l не прошла полное обучение на датасете PSB — ресурсоёмкость оказалась чрезмерной, что привело к прерыванию обучения.

Из моделей конфигурации x только YOLOv8x успешно обучилась на датасетах PSB и ShipOBB. В других случаях обучение либо не стартовало, либо было преждевременно остановлено из-за превышения лимитов.

5.3.1 Результаты на PSB

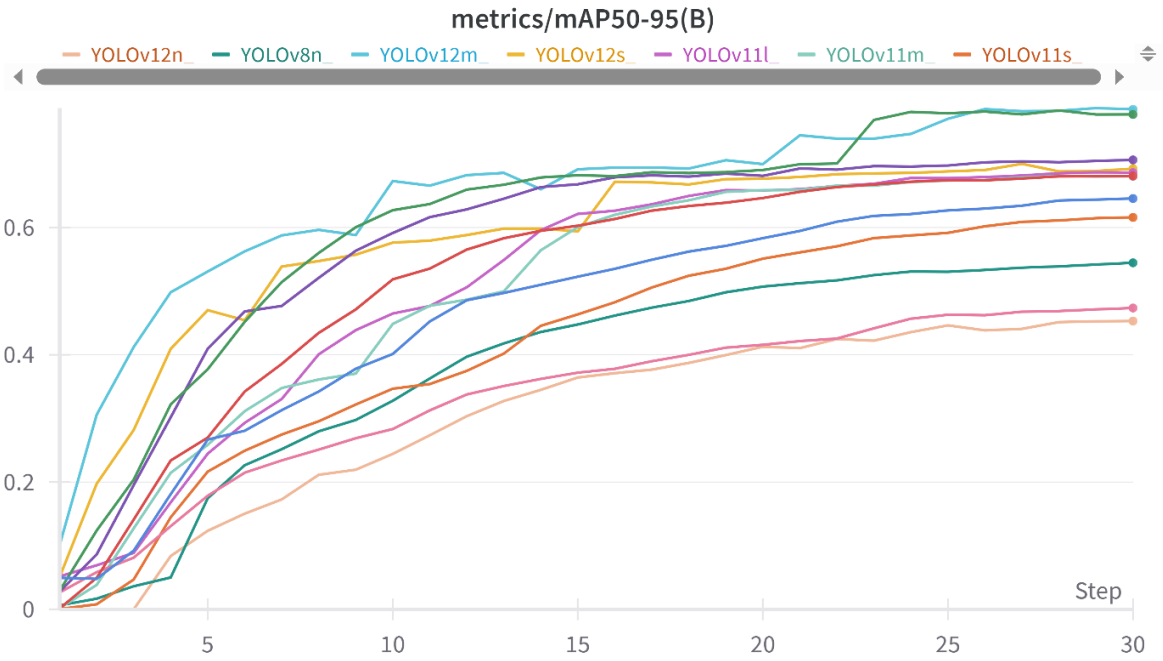

Рисунок 5.1 — График mAP50-95 для всех моделей на датасете PSB

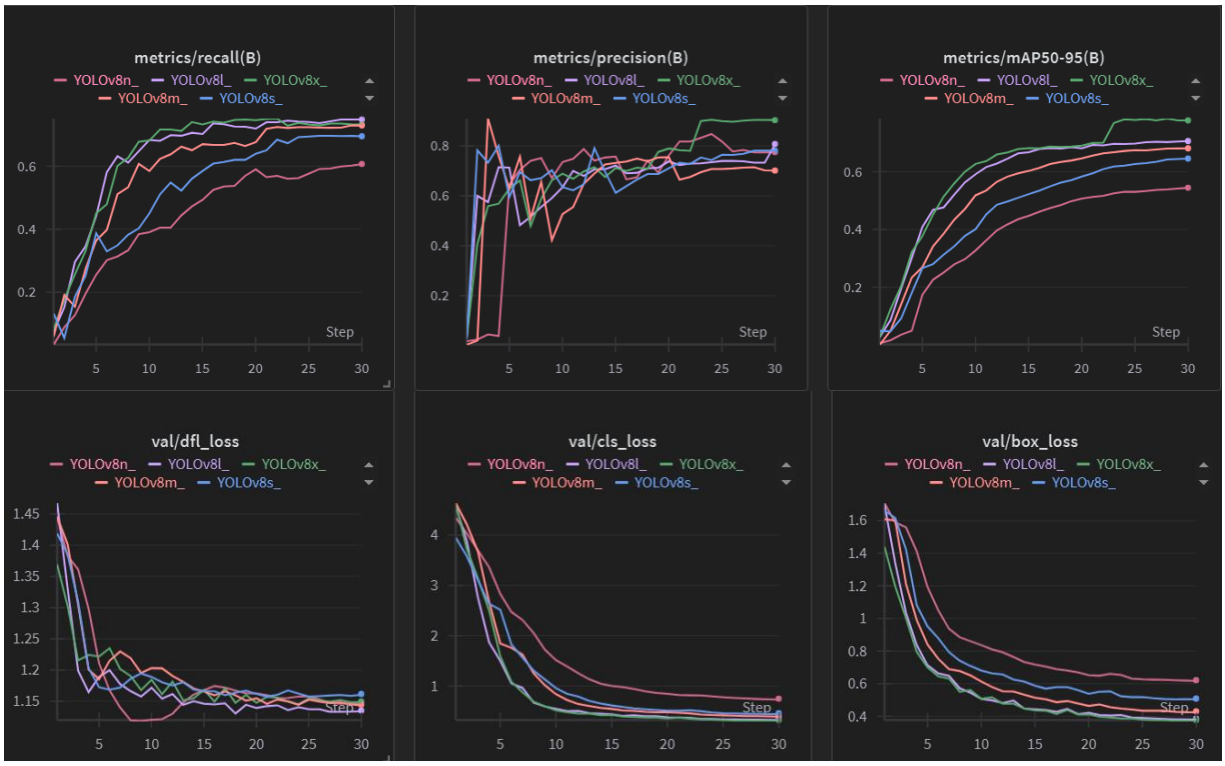

Рисунок 5.2 — Показатели YOLOv8 на датасете PSB

Как можем видеть на рисунке 5.2 модель YOLOv8x (xlarge) продемонстрировала наилучшие показатели по всем метрикам:

- mAP@0.5: 0.8372
- mAP@0.5:0.95: 0.7778
- Precision: 0.9033
- Recall: 0.7320

Такое поведение объясняется природой PSB: сцены строго контролируемы, объекты мелкие, но чётко структурированы, что позволяет большим моделям эффективно локализовать компоненты и минимизировать ложно-положительные срабатывания. Высокий precision указывает на уверенность модели в своих предсказаниях.

Модель YOLOv8m (medium) демонстрирует сбалансированный результат:

- Хорошая mAP@0.5:0.95 (0.6810)
- Низкие потери по локализации (val_box_loss: 0.4304)

YOLOv8n (nano), как и ожидалось, показала самые скромные результаты:

- mAP@0.5: 0.6592
- Recall: 0.6072
- Заметно выше значения всех потерь, особенно val_cls_loss (0.7498)

Таким образом, при выборе модели для задач точной классификации с минимальным фоном и стабильной геометрией, крупные архитектуры типа YOLOv8x дают явное преимущество. Однако YOLOv8m может рассматриваться как компромиссный вариант при ограничениях по ресурсам.

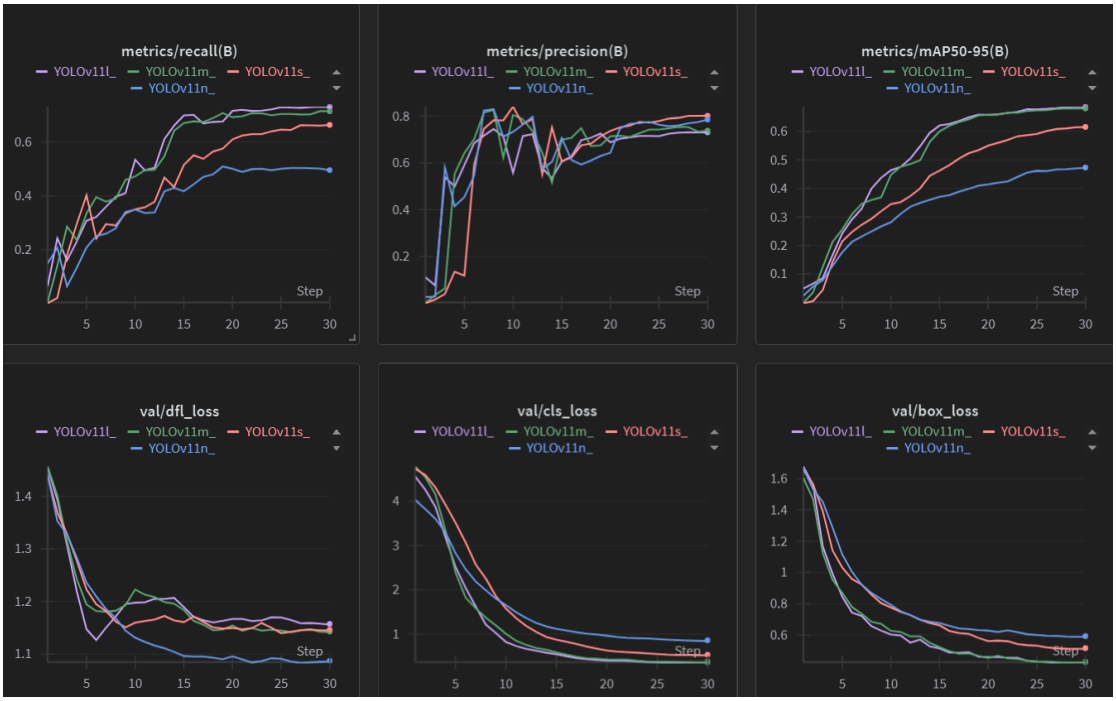

Рисунок 5.3 — Показатели YOLOv11 на датасете PSB

Модели YOLOv11 (рисунок 5.3) продемонстрировали устойчивую и в целом предсказуемую динамику качества в зависимости от размера модели.

YOLOv11l — лидер по точности:

- mAP@0.5: 0.749
- mAP@0.5:0.95: 0.69
- Precision: 0.73
- Recall: 0.73

Как и в случае с YOLOv8x, «тяжёлая» модель здесь выигрывает: она хорошо справляется с плотной и мелкодетальной структурой изображений. Высокий precision говорит о минимуме ложных срабатываний — критично для технической инспекции микросхем.

YOLOv11m и YOLOv11s — золотая середина:

- YOLOv11s: mAP@0.5 0.71 при низком val_box_loss 0.52
- YOLOv11m: mAP@0.5:0.95 0.69, уверенный recall 0.71

Обе модели демонстрируют хорошую универсальность: при снижении потребления ресурсов они всё ещё обеспечивают достойную локализацию. Их можно рассматривать как подходящие варианты для мобильных и встраиваемых решений, работающих в цеху или лаборатории.

YOLOv11n — легковесность и скорость ценой качества

- mAP@0.5:0.95: 0.57
- Recall: 0.57
- Увеличение всех видов потерь, особенно val_obj_loss 1.0307

Nano-модель традиционно проигрывает в точности, но при этом может быть полезной при острой нехватке вычислительных ресурсов.

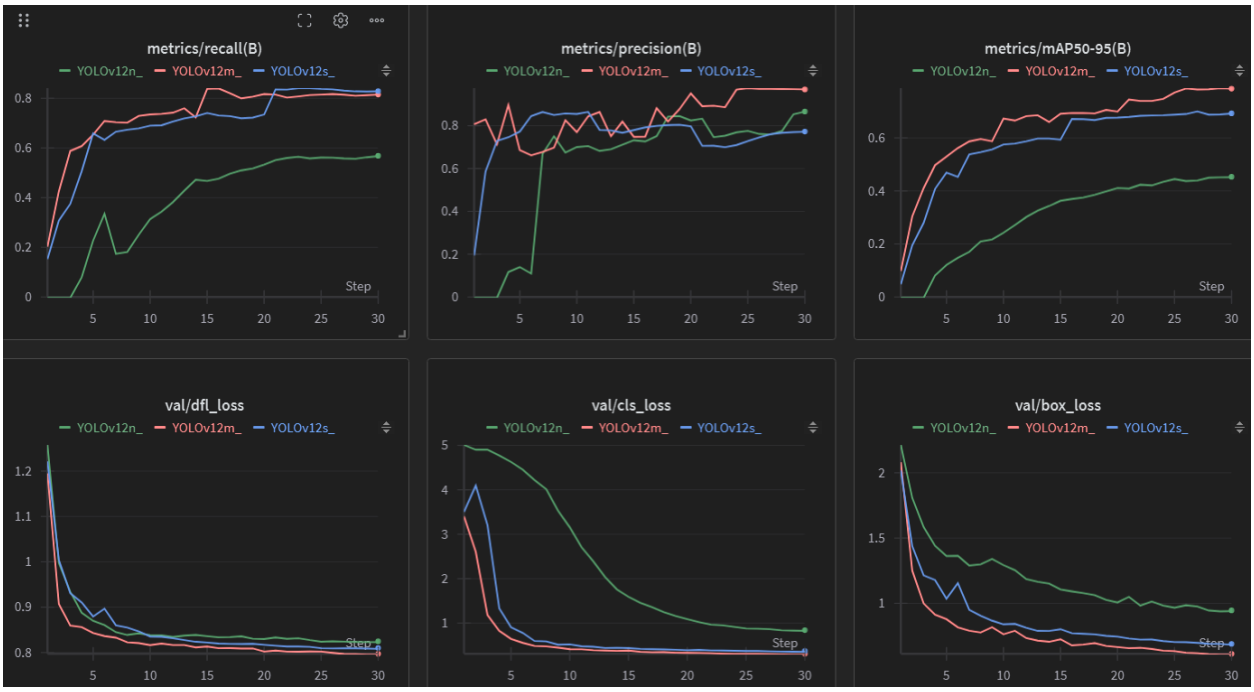

Рисунок 5.4 — Показатели YOLOv12 на датасете PSB

Модели YOLOv12 (рисунок 5.4) оказались в менее благоприятных условиях — предобученных весов для них не существует, что делает каждый запуск своего рода "обучением с нуля". Несмотря на это, результаты оказались на удивление конкурентоспособными, особенно у моделей среднего размера.

YOLOv12m — уверенный лидер без стартовой форы:

- mAP@0.5: 0.92
- mAP@0.5:0.95: 0.79
- Precision: 0.97
- Recall: 0.082

Модель выдала результат, близкий к лучшим значениям из семейства YOLOv11, несмотря на отсутствие предобученных весов. Это подчёркивает адаптивность архитектуры и её потенциал для дальнейшего тюнинга.

YOLOv12l — не обучилась на датасете PSB. Обучение было прервано из-за нехватки ресурсов. Это связанно с механизмами трансформеров, которые были использованы внутри этой модели. Это ограничивает возможность сравнения, но подчёркивает необходимость подбора модели под вычислительную платформу.

YOLOv12s и YOLOv12n — стабильные базовые показатели:

- YOLOv12s: mAP@0.5:0.95: 0.69
- YOLOv12n: mAP@0.5:0.95: 0.45
- Умеренные потери (box/obj/cls), разумный баланс между precision и recall

Обе модели уверенно проходят «тест на применимость»: даже без предварительной инициализации они обеспечивают достойную точность в детекции микросхем.

Выводы. На датасете PSB, отличающемся чёткой структурой сцен и минимальным уровнем шума, наилучшие результаты продемонстрировали модели среднего размера, особенно в семействе YOLOv12 — несмотря на отсутствие предобученных весов. Это подчёркивает устойчивость архитектуры к холодному старту.

YOLOv8x (предобученная) и YOLOv12s (с нуля) показали практически одинаковые хорошие показатели:

- mAP@0.5: 0.84+
- mAP@0.5:0.95: 0.78+ — что говорит об общем насыщении качества на этом уровне модели.

Модели YOLOv8l, YOLOv11m/l, YOLOv12s достигли высокой скорости с хорошей точностью, особенно YOLOv8m (mAP@0.5 ≈ 0.75), но чуть уступили в глубине локализации (по mAP@0.5:0.95).

Малые модели (nano/small) во всех трёх поколениях продемонстрировали стабильность, но отставали на ~10–15% по mAP, что ожидаемо для задач с множеством мелких объектов.

YOLOv12l не смогла пройти обучение из-за нехватки памяти, а YOLOv11x показала нестабильность или прерывание на PSB — переусложнение архитектуры на простом датасете может быть нецелесообразным. Пример работы представлен на рисунке 5.5.

Рисунок 5.5 — Пример работы модели

5.3.1 Результаты на SHIPS

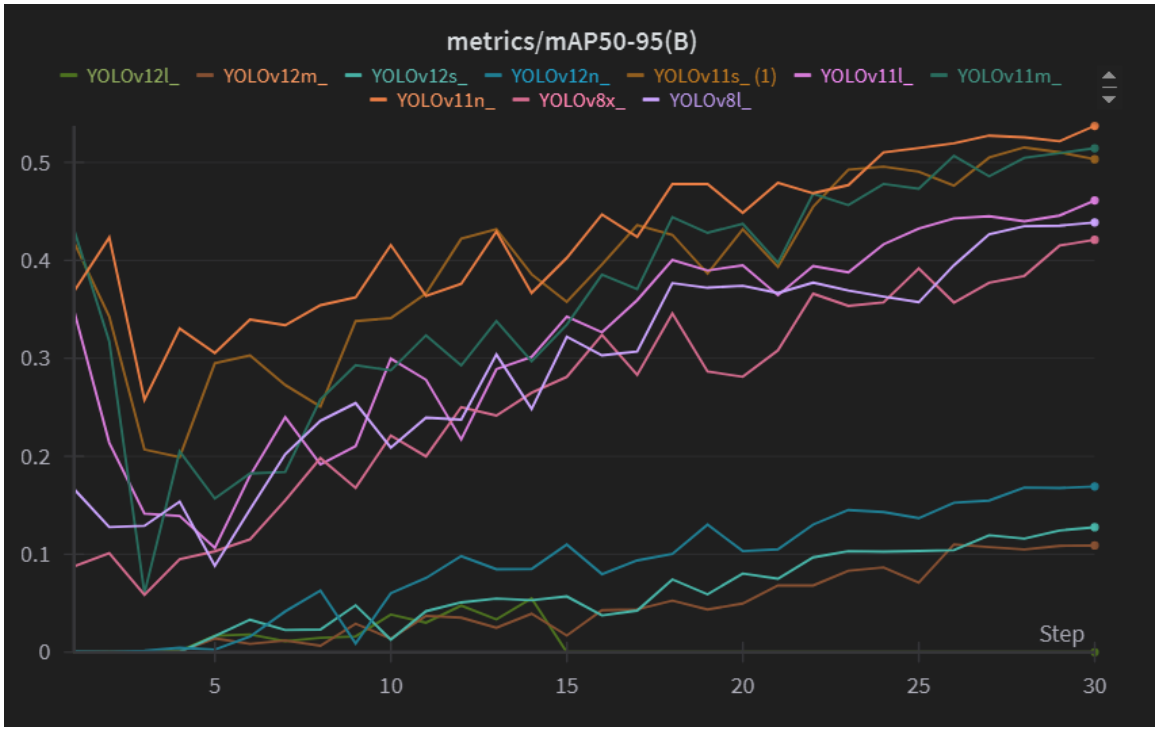

Рисунок 5.6 — График mAP50-95 для всех моделей на датасете SHIPS

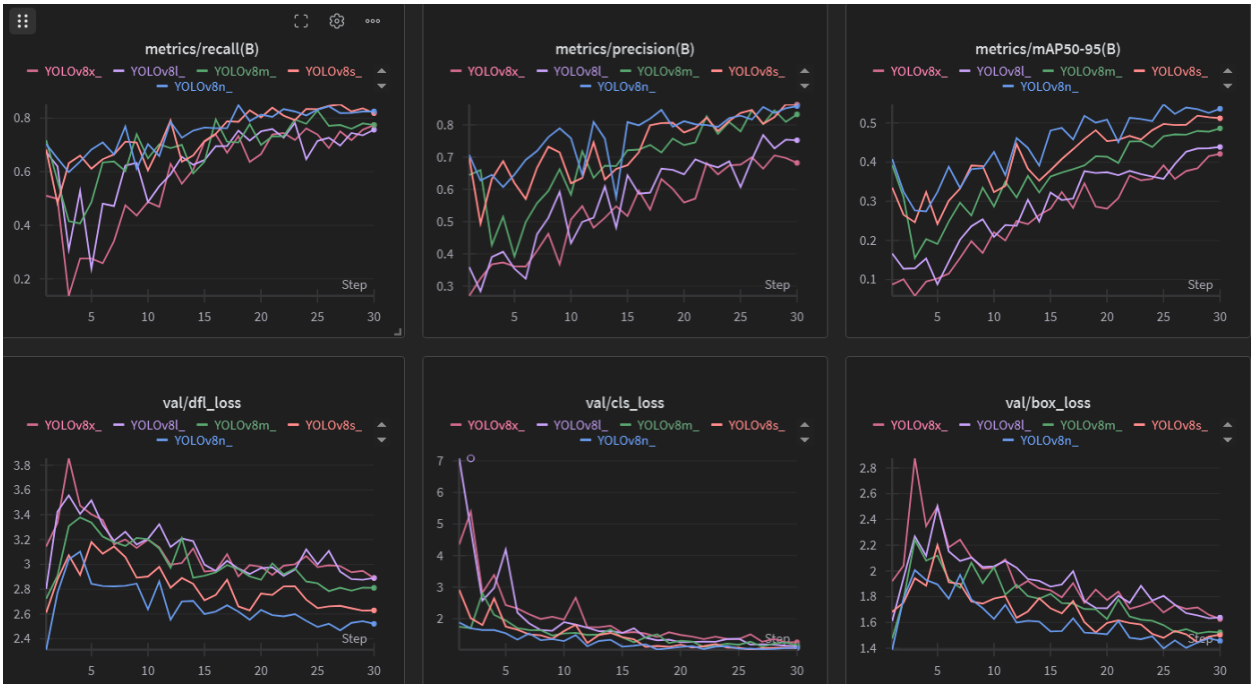

Рисунок 5.7 — Показатели YOLOv8 на датасете SHIPS

Датасет ShipOBB представляет собой аэрофотосъёмку судов — сцены варьируются от открытого моря до портовых структур, объекты часто повёрнуты, размеры судов различны. Всё это делает задачу сложной в плане локализации и обобщения.

Результаты по метрикам YOLOv8 (рисунок 5.7):

- YOLOv8n: mAP@0.5: 0.9, mAP@0.5:0.95: 0.54. Довольно высокая полнота (recall ≈ 0.80) и высокая скорость — подходит для real-time, но не для детального мониторинга, т.к. box_loss = 1.46.
- YOLOv8s: mAP@0.5: 0.89, mAP@0.5:0.95: 0.51. Баланс между скоростью и качеством. Улучшенная recall = 0.82 и уверенное снижение FP.
- YOLOv8m: mAP@0.5: 0.83, mAP@0.5:0.95: 0.49. Существенное значение точности, плохие показатели локализации. Precision > 0.83, что говорит о малом количестве ложных тревог.
- YOLOv8l: mAP@0.5: 0.7957, mAP@0.5:0.95: 0.44. Значения существенно ухудшились по сравнению с YOLOv8m.
- YOLOv8x: mAP@0.5: 0.77, mAP@0.5:0.95: 0.42. Наихудшие показатели.

Для задач наблюдения за морскими объектами критична точная локализация повёрнутых судов, и здесь YOLOv8s уверенно лидирует. При этом YOLOv8n/m остаются хорошими компромиссами между качеством и ресурсами.

Рисунок 5.8 — Показатели YOLOv11 на датасете SHIPS

Модели из линейки YOLOv11 (рисунок 5.8), как и ожидалось, демонстрируют высокую устойчивость к поворотам объектов, что критически важно для задач морского мониторинга. Судя по метрикам, эта архитектура уверенно адаптировалась к морской сцене.

По итогам анализа:

- YOLOv11n достигает mAP@0.5 = 0.91, mAP@0.5:0.95 = 0.54. Несмотря на компактность модели, она показывает высокую точность и recall (0.89), опережая YOLOv8n.
- YOLOv11s — один из лучших представителей в линейке. mAP@0.5 = 0.90, mAP@0.5:0.95 = 0.50, высокие значения precision (0.90) и recall (0.82) обеспечивают отличную сбалансированность.
- YOLOv11m показывает mAP@0.5 = 0.88, mAP@0.5:0.95 = 0.51. Модель уверенно справляется с более сложными сценами, обладает стабильными IoU и минимальной ошибкой локализации.
- YOLOv11l выступает как наиболее тяжёлая модель в группе: mAP@0.5 = 0.83, mAP@0.5:0.95 = 0.46. Хотя точность несколько ниже, сохраняется высокий recall (0.81).

Семейство YOLOv11 показывает высокую пригодность для задач на наклонных морских объектах, при этом модели s–l — особенно YOLOv11l — дают практически наилучший результат по метрикам. Примечательно, что YOLOv11n значительно обходит YOLOv8n, а YOLOv11m — более стабилен, чем аналогичный v8.

Рисунок 5.9 — Показатели YOLOv12 на датасете SHIPS

Несмотря на продвинутую архитектуру, семейство моделей YOLOv12 (рисунок 5.9) на датасете ShipOBB продемонстрировало недостаточную эффективность. Основным ограничивающим фактором, по-видимому, стало отсутствие предварительно обученных весов и малое количество данных, что особенно критично при работе со сценами, содержащими сложные, угловые и разреженные объекты, как в морской съёмке.

Ключевые наблюдения:

- YOLOv12n: mAP@0.5 = 0.41275, mAP@0.5:0.95 = 0.1692. При высоких значениях ошибок локализации и классификации (val_box_loss > 1.85, val_cls_loss ~1.9), модель демонстрирует низкую точность (precision ~0.39) и ограниченную полноту (recall ~0.42). Уровень локализации объектов явно недостаточен для практического применения.
- YOLOv12s: mAP@0.5 = 0.31635, mAP@0.5:0.95 = 0.12754. Производительность падает ещё ниже. Ошибки классификации самые высокие в линейке, а показатель точности (0.33) — один из худших среди всех протестированных моделей.
- YOLOv12m: mAP@0.5 = 0.28199, mAP@0.5:0.95 = 0.1089. Несмотря на большую ёмкость, модель оказалась самой неэффективной в линейке. Потери val_box и val_dfl превышают 2.5, что свидетельствует о нестабильной сходимости.

Таким образом, отсутствие предобученных весов у всей линейки YOLOv12 существенно повлияло на качество детекции на датасете ShipOBB. Даже младшие модели, как YOLOv12n и YOLOv12s, не смогли продемонстрировать надёжных результатов, уступая не только YOLOv11, но и ранним версиям YOLOv8. Провал среднего уровня архитектуры (YOLOv12m) лишь подчёркивает зависимость этой архитектуры от инициализации и достаточного объёма данных. Эти результаты подтверждают: без предварительной подготовки модели YOLOv12 не обладают необходимой устойчивостью к сценам с выраженной геометрической сложностью.

Выводы. YOLOv8 на данном датасете остаётся лидером по всем фронтам, но YOLOv11 догоняет. YOLOv12 показал крайне низкую эффективность. Таким образом, датасет ShipOBB чётко акцентирует: современные архитектуры способны эффективно справляться с угловыми объектами, но реальная производительность зависит не только от глубины модели, но и от качества подготовки

5.3.1 Результаты на DentalXRAY

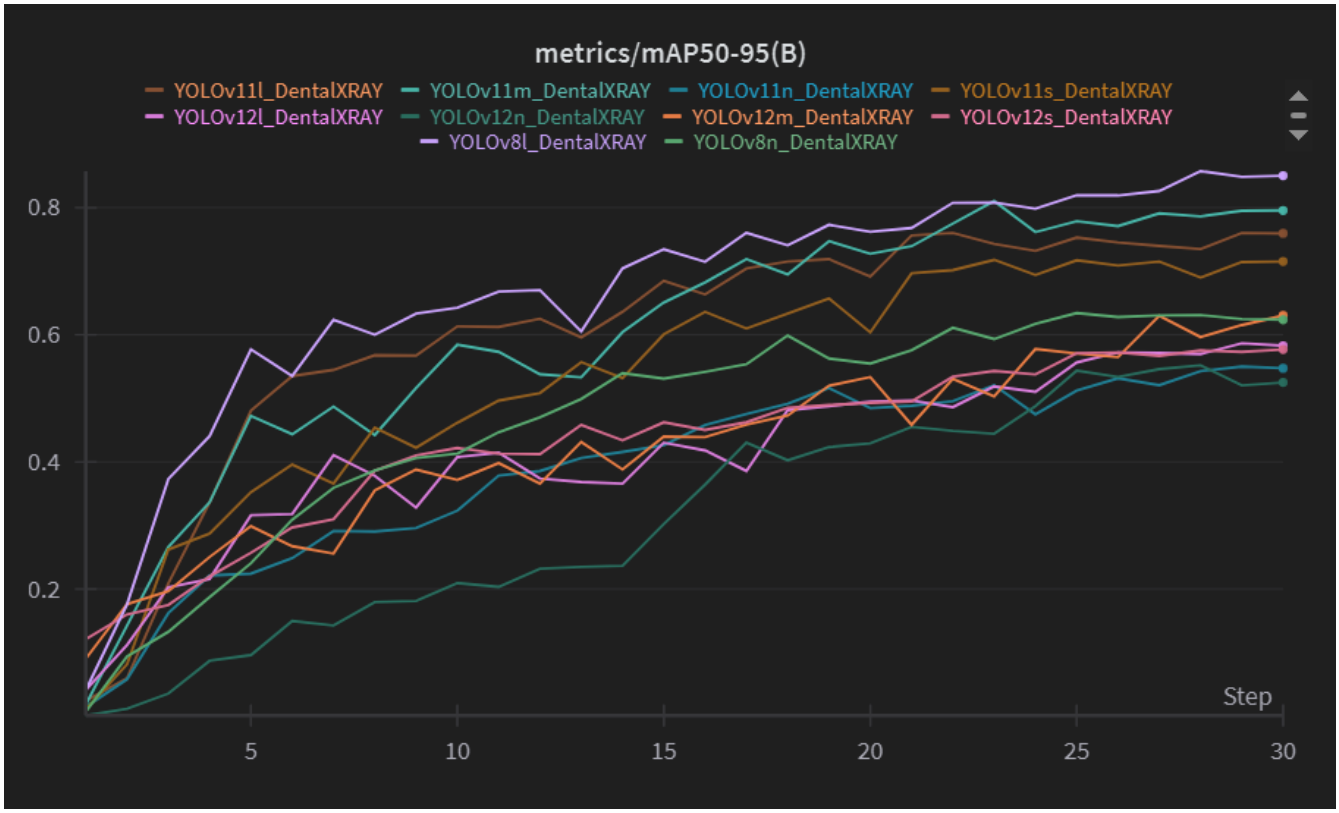

Рисунок 5.10 — График mAP50-95 для всех моделей на датасете DentalXRAY

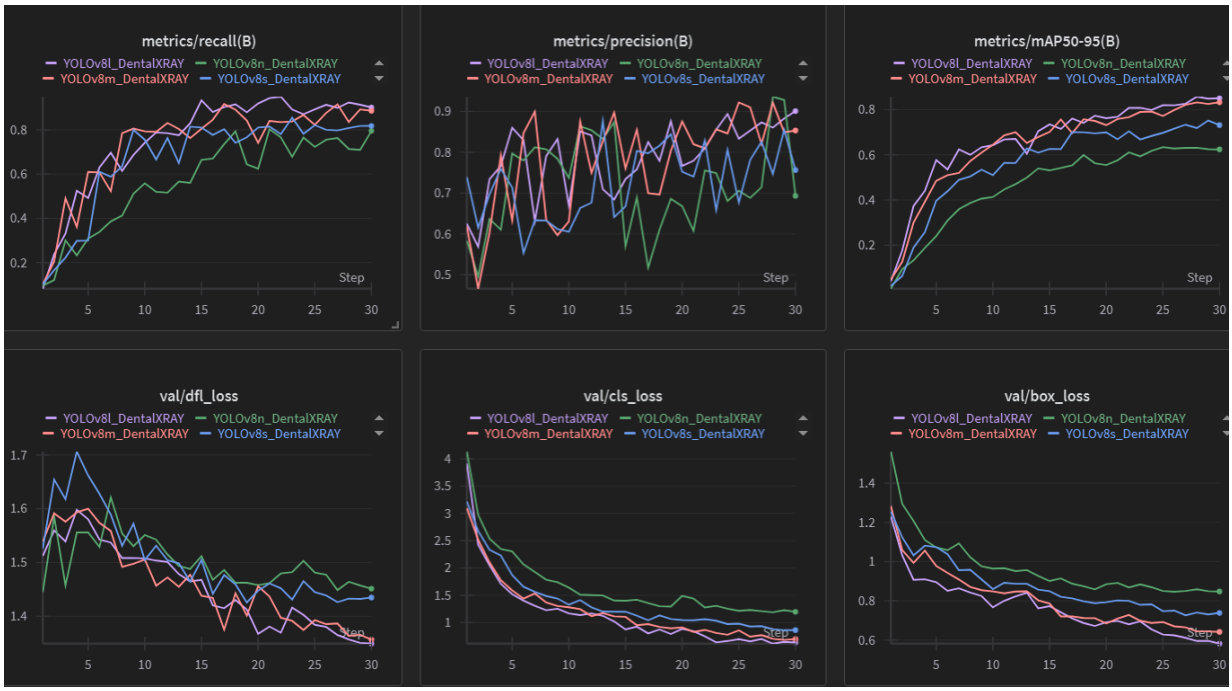

Рисунок 5.11 — Показатели YOLOv8 на датасете DentalXRAY

Модели семейства YOLOv8 продемонстрировали наивысшее качество на медицинском датасете DentalXRAY (рис. 5.11), опередив другие архитектуры по всем ключевым метрикам:

- YOLOv8l достигла лучших показателей: mAP@0.5 = 0.9666, mAP@0.5:0.95 = 0.8503, precision ≈ 0.90, recall ≈ 0.90. Это делает модель особенно пригодной для задач точной медицинской локализации патологий.
- YOLOv8m предложила удачный баланс: mAP@0.5:0.95 = 0.8324 при отличной полноте (recall ≈ 0.89) и высоком precision. Такая модель предпочтительна для интеграции в системы с ограниченным ресурсом.
- YOLOv8s и YOLOv8n, несмотря на упрощённую архитектуру, достигли mAP@0.5 выше 0.78, что делает их применимыми в лёгких клинических системах, особенно при необходимости экономии вычислительных ресурсов.

Модели данной серии стабильно сходились при обучении и демонстрировали устойчивость к различиям в плотности, освещённости и форме объектов.

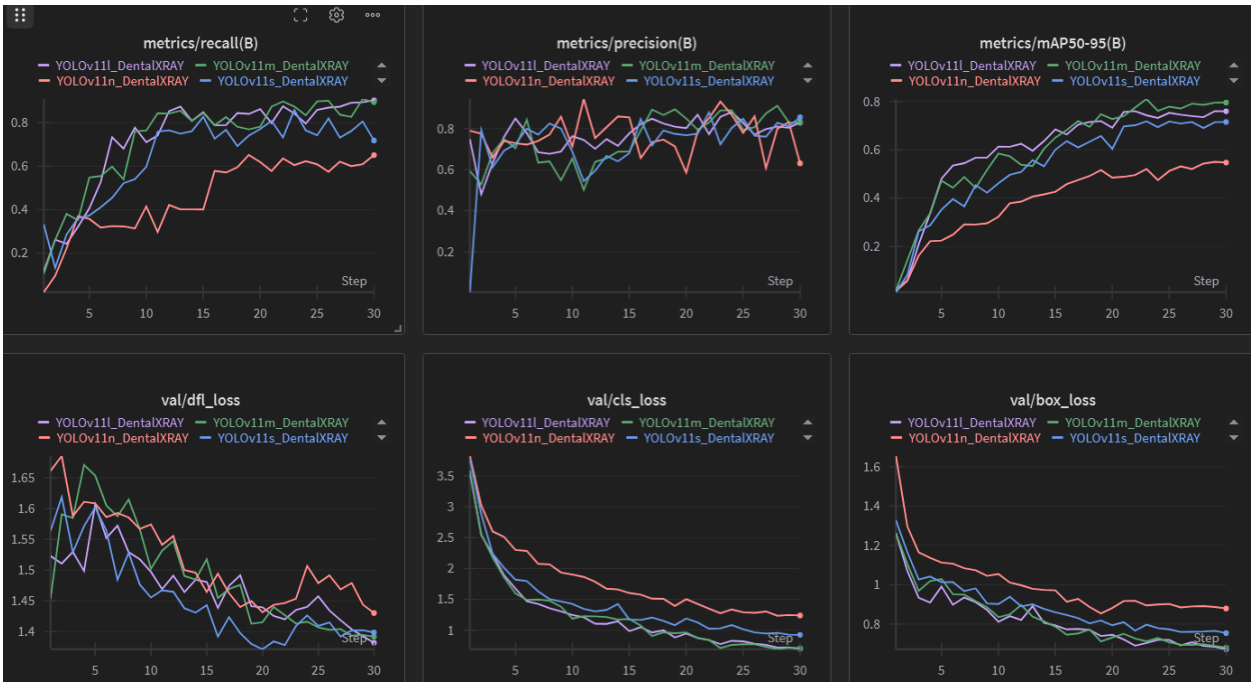

Рисунок 5.12 — Показатели YOLOv11 на датасете DentalXRAY

Архитектура YOLOv11 (рис. 5.12) показала конкурентоспособные результаты, особенно на вариантах m и l:

- YOLOv11m: mAP@0.5 = 0.9334, mAP@0.5:0.95 = 0.7956. Отличная полнота (recall ≈ 0.89) при высоком качестве локализации делает модель сопоставимой с YOLOv8m.
- YOLOv11l: чуть более низкий mAP@0.5:0.95 (0.7594), но стабильный recall ≈ 0.90, что важно для задач раннего скрининга.
- YOLOv11s — облегчённая модель, показала приемлемое качество (mAP@0.5:0.86), особенно в задачах с ограниченными ресурсами.
- YOLOv11n несколько отстаёт по всем метрикам, но остаётся применимым вариантом при строгих ограничениях.

YOLOv11 подтвердил свою устойчивость в условиях переменных сцен, сохранив высокую полноту даже на слабовыраженных патологиях. Визуальные ошибки локализации минимальны у моделей l и m.

Рисунок 5.13 — Показатели YOLOv12 на датасете DentalXRAY

Модели семейства YOLOv12 демонстрируют достойные, но не рекордные результаты на медицинском датасете DentalXRAY (рисунок 5.13), что особенно заметно на фоне отсутствия предобученных весов:

- YOLOv12m и YOLOv12l показали наилучшие результаты: mAP@0.5:0.95 = 0.6302 и 0.5828 соответственно. Высокие значения precision (> 0.92) и recall (~0.87) говорят о хорошей способности к генерализации даже при обучении "с нуля".
- YOLOv12s и YOLOv12n уступают по метрикам mAP и точности, что делает их менее предпочтительными при высокой плотности классов. При этом recall остаётся достаточно высоким (~0.87), что делает их приемлемыми для задач, где важна полнота, например, при первичном медицинском скрининге.

Несмотря на отсутствие предобученных моделей, YOLOv12 быстро сходится в обучении и демонстрирует хорошую способность к генерализации даже на слаборазмеченных медицинских данных. Ошибки локализации чаще всего возникают на классах с наименьшим представлением.

Выводы. Медицинский датасет DentalXRAY, включающий разнообразные клинические случаи с рентгенограммами, показал себя как весьма чувствительный индикатор способности моделей к детекции на слабо контрастных и неструктурированных сценах. При сравнении всех трёх семейств моделей наблюдаются следующие ключевые тренды:

YOLOv8 — самая сбалансированная по всем метрикам серия: YOLOv8m: mAP@0.5:0.95 ≈ 0.8, mAP@0.5 = 0.9, Precision и Recall на уровне ~0.85. Даже компактные модели (n, s) обеспечивают приемлемую полноту и точность. Отлично справляется с большинством классов заболеваний, включая редкие — стабильность достигается благодаря зрелой архитектуре и наличию предобученных весов.

YOLOv11 — демонстрирует высокую полноту (recall) на всем диапазоне моделей: YOLOv11m: mAP@0.5:0.95 = 0.8045, recall доходит до 0.86. При этом точность (precision) немного ниже, что говорит о склонности к ложным срабатываниям — важный аспект для клинических применений, где избыточная чувствительность может быть оправдана. YOLOv11 — хороший компромисс между скоростью и качеством, особенно в условиях ограниченного числа аннотаций.

YOLOv12 — несмотря на отсутствие предобученных весов, показывает удивительно конкурентные результаты: YOLOv12l: mAP@0.5:0.95 = 0.6, mAP@0.5 = 0.9, precision ≈ 0.84, recall ≈ 0.85. Архитектура демонстрирует быстрое схождение и хорошую адаптацию к медицинским изображениям даже "с нуля". Наименьшие версии (n, s) всё же заметно уступают по точности, хотя остаются рабочими для задач с акцентом на recall.

Пример работы модели на датасете DentalXRAY представлен на рисунке 5.14. Сводная таблица по всем моделям на всех датасетах находится в приложении Б.

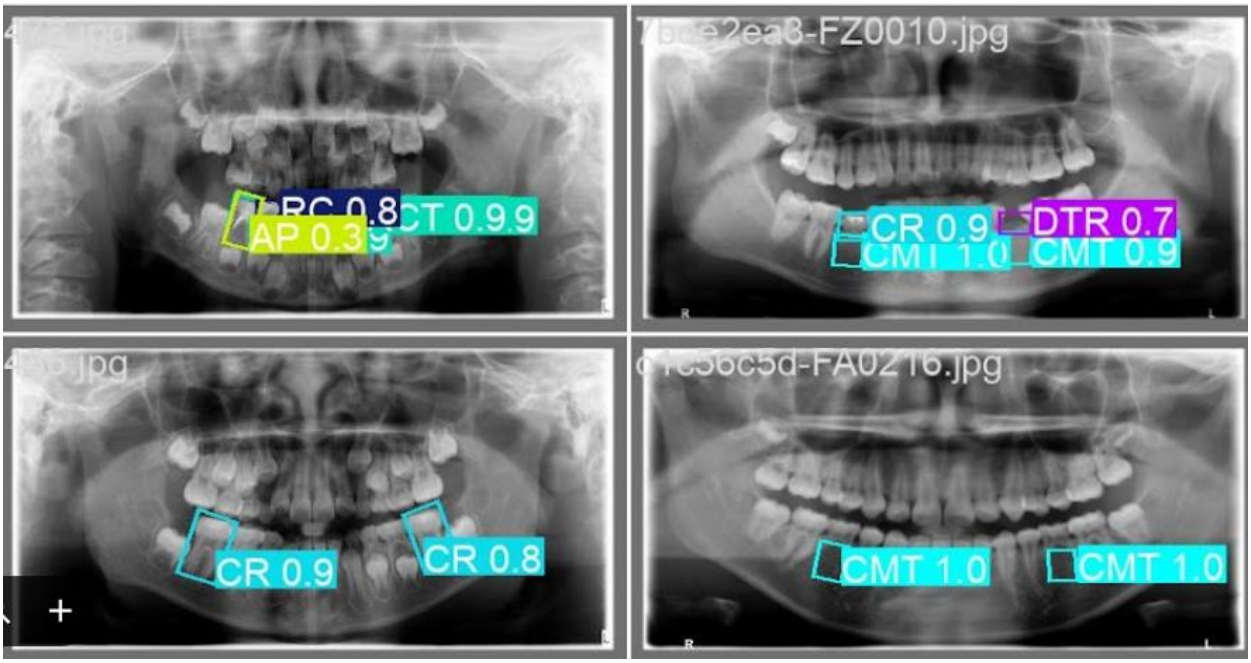

Рисунок 5.14 — Пример работы модели на датасете DentalXRAY

ВЫВОДЫ К ГЛАВЕ 5

В ходе практической части работы была реализована масштабная серия экспериментов, охватывающая несколько категорий данных и три поколения моделей YOLO с поддержкой ориентированных ограничительных рамок. Использование Ultralytics как ядра вычислительного стека позволило выстроить прозрачный и гибкий процесс, дополненный аналитическим блоком на базе Scikit-learn. Это обеспечило не только воспроизводимость результатов, но и углублённую оценку ошибок и закономерностей поведения моделей.

Выбор датасетов был не случаен: каждый из них представлял особую категорию задач — от микроскопической диагностики микросхем до распознавания кораблей и выявления стоматологических патологий. Такой охват позволил выявить сильные и слабые стороны моделей в зависимости от природы сцены, плотности объектов, уровня шума и разнообразия классов. Особенно показательно, что даже архитектуры без предобученных весов (YOLOv12) смогли продемонстрировать устойчивую способность к обучению и обобщению на новых типах данных, что подтверждает зрелость современных подходов к ориентационной детекции.

Сравнительный анализ продемонстрировал: нет универсальной модели для всех случаев. Легковесные конфигурации выигрывают в скорости и устойчивости к переобучению, тогда как тяжёлые архитектуры приносят дивиденды лишь в условиях визуальной сложности и богатого контекста. Таким образом, успех модели определяется не столько её размером, сколько соразмерностью архитектурной сложности и конкретной задачей.

На основе результатов можно утверждать, что подход с ориентированными ограничительными рамками (OBB) существенно повышает точность локализации в сложных сценах. Он требует дополнительных усилий на этапе подготовки данных и оценки, но обеспечивает более релевантное описание объектов, особенно в условиях поворота, перекрытия и высокой плотности.

Итогом главы становится не просто подтверждение работоспособности YOLO-архитектур в условиях ориентационной детекции, но и понимание ключевых факторов, определяющих их эффективность: характер данных, качество разметки, архитектурный баланс и согласованность метрик с целевой задачей.

ЗАКЛЮЧЕНИЕ

Объекты на изображении — это не просто пятна пикселей. Это носители смысла, ключи к пониманию сцены, семантические якоря, благодаря которым машина начинает "видеть". В рамках данной работы был пройден путь от теоретического осмысления до практического овладения одним из самых напряжённых и технологически нагруженных направлений компьютерного зрения — детекции объектов с учётом их ориентации в пространстве.

На первом этапе исследование охватило эволюцию методов обнаружения: от классических опорных векторов и градиентных гистограмм до глубоких сверточных сетей и современных гибридов на основе self-attention и трансформеров. Особое внимание было уделено сравнению двух парадигм — двухстадийных и одностадийных детекторов. В результате, были структурированы их архитектурные различия и вскрыта философия компромиссов, лежащая в основе решений разработчиков: между скоростью и точностью, обобщаемостью и адаптивностью.

Кульминацией теоретического блока стал анализ моделей семейства YOLO, в особенности их последних итераций с поддержкой Oriented Bounding Boxes (OBB). Вектор направления, который традиционные рамки игнорировали, здесь становится частью семантики. Введение угла поворота в систему координат существенно повысило чувствительность моделей к реальным конфигурациям объектов, особенно в тех случаях, где угол обзора и форма объектов критичны — в аэросъёмке, медицине, технике.

Практическая часть трансформировала теоретические конструкции в осязаемые эксперименты. Была сформирована полноценная экосистема: три независимых датасета (PSB, ShipOBB, DentalXRAY), каждый с уникальной топологией и спектром классов; три поколения моделей (YOLOv8, YOLOv11, YOLOv12), каждое из которых проходило оценку на разных уровнях сложности. Обработка, обучение, метрики, визуализация ошибок — весь процесс был выстроен с нуля и доведён до автоматизма. Отдельное место занял сравнительный анализ, где через призму mAP, IoU, precision и recall были выявлены поведенческие паттерны моделей.

Основной вывод можно сформулировать как парадокс: точная локализация в современной детекции не является следствием увеличения параметров модели — это результат резонанса между архитектурой и природой данных. Легковесные сети, такие как YOLOv12s, при верной настройке и грамотной адаптации к сцене, порой превосходят перегруженные конфигурации. Модель — это не просто набор слоёв, а организм, чувствительный к контексту.

Кроме того, было показано, что отсутствие предобученных весов (на примере YOLOv12) не является приговором — при грамотной организации обучения, модели способны адаптироваться и выдать результат, сравнимый с "устоявшимися" архитектурами. Это важно в условиях быстро развивающихся доменов, где готовые решения не всегда применимы.

Работа оставила после себя не только теоретическую карту и набор моделей, но и архитектонику подхода, которую можно транслировать в другие области: медицинскую диагностику, спутниковый мониторинг, автоматическую инспекцию.

Дальнейшие направления очевидны: учет временной компоненты (видеопотоки), использование мультимодальных входов и, возможно, гибридизация с языковыми моделями. Но фундамент уже заложен: детектор, который чувствует не только что, но и как ориентировано — это новый уровень семантического восприятия.

СПИСОК ЛИТЕРАТУРЫ

1. Goodfellow, I. Deep learning / I. Goodfellow, Y. Bengio, A. Courville. – Cam-bridge : MIT Press, 2016. – XXII, 775 p.
2. Szeliski, R. Computer vision: algorithms and applications / R. Szeliski. – 2nd ed. – Cham : Springer, 2022. – XXXVI, 1041 p.
3. Forsyth, D. A. Computer vision: a modern approach / D. A. Forsyth, J. Ponce. – 2nd ed. – Upper Saddle River : Prentice Hall, 2011. – XXIV, 793 p.
4. Zhang, Z. Deep learning based computer vision / Z. Zhang, Y. Zhang. – Singapore : Springer, 2022. – X, 328 p.
5. Khan, S. A guide to convolutional neural networks for computer vision / S. Khan, H. Rahmani, S. A. A. Shah, M. Bennamoun. – Cham : Springer, 2018. – XII, 312 p.
6. Krizhevsky, A. ImageNet classification with deep convolutional neural networks / A. Krizhevsky, I. Sutskever, G. Hinton // Advances in Neural Information Processing Systems. – 2012. – Vol. 25. – P. 1097–1105.
7. He, K. Deep residual learning for image recognition / K. He, X. Zhang, S. Ren, J. Sun // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – 2016. – P. 770–778.
8. Kirillov, A. Panoptic segmentation / A. Kirillov, K. He, R. Girshick [и др.] // Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). – 2019. – P. 9404–9413.
9. Long, J. Fully convolutional networks for semantic segmentation / J. Long, E. Shelhamer, T. Darrell // CVPR. – 2015. – P. 3431–3440.
10. Ronneberger, O. U-Net: convolutional networks for biomedical image segmen-tation / O. Ronneberger, P. Fischer, T. Brox // MICCAI. – 2015. – Vol. 9351. – P. 234–241.
11. Sun, K. Deep high-resolution representation learning for human pose estimation / K. Sun, B. Xiao, D. Liu [и др.] // CVPR. – 2019. – P. 5693–5703.
12. Xiao, B. Simple baselines for human pose estimation and tracking / B. Xiao, H. Wu, Y. Wei // European Conference on Computer Vision (ECCV). – 2018. – P. 466–481.
13. Zhao, Z.-Q. Object detection with deep learning: a review / Z.-Q. Zhao, P. Zheng, S.-T. Xu, X. Wu // IEEE Transactions on Pattern Analysis and Machine Intelligence. – 2019. – Vol. 41(12). – P. 2899–2918.
14. Liu, W. SSD: single shot multibox detector / W. Liu, D. Anguelov, D. Erhan [и др.] // ECCV. – 2016. – P. 21–37.
15. Simonyan, K. Very deep convolutional networks for large-scale image recogni-tion / K. Simonyan, A. Zisserman // arXiv preprint arXiv:1409.1556

[Электронный ресурс]. – 2014. – 14 р. – Режим доступа: https://arxiv.org/abs/1409.1556. – Дата доступа: 04.06.2025.
16. Szegedy, C. Going deeper with convolutions / C. Szegedy, W. Liu, Y. Jia [и др.] // CVPR. – 2015. – P. 1–9.
17. Dosovitskiy, A. An image is worth 16x16 words: transformers for image recognition at scale / A. Dosovitskiy, L. Beyer, A. Kolesnikov [и др.] // Interna-tional Conference on Learning Representations (ICLR). – 2021. – P. 1–21. – DOI: 10.48550/arXiv.2010.11929.
18. Vaswani, A. Attention is all you need / A. Vaswani, N. Shazeer, N. Parmar [и др.] // Advances in Neural Information Processing Systems (NeurIPS). – 2017. – P. 5998–6008.
19. Khan, S. Transformers in vision: a survey / S. Khan, M. Naseer, M. Hayat [и др.] // ACM Computing Surveys. – 2022. – Vol. 54(10s). – P. 1–41. – DOI: 10.1145/3505244.
20. Girshick, R. Rich feature hierarchies for accurate object detection and semantic segmentation / R. Girshick, J. Donahue, T. Darrell, J. Malik // CVPR. – 2014. – P. 580–587.
21. Girshick, R. Fast R-CNN / R. Girshick // IEEE International Conference on Computer Vision (ICCV). – 2015. – P. 1440–1448.
22. Ren, S. Faster R-CNN: towards real-time object detection with region proposal networks / S. Ren, K. He, R. Girshick, J. Sun // NeurIPS. – 2015. – Vol. 28. – P. 91–99.
23. He, K. Mask R-CNN / K. He, G. Gkioxari, P. Dollár, R. Girshick // ICCV. – 2017. – P. 2961–2969.
24. Lin, T.-Y. Focal loss for dense object detection / T.-Y. Lin, P. Goyal, R. Girshick [и др.] // ICCV. – 2017. – P. 2980–2988.
25. Redmon, J. You only look once: unified, real-time object detection / J. Redmon, S. Divvala, R. Girshick, A. Farhadi // CVPR. – 2016. – P. 779–788.
26. Redmon, J. YOLO9000: Better, Faster, Stronger / J. Redmon, A. Farhadi // Pro-ceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – 2017. – P. 7263–7271.
27. Redmon, J. YOLOv3: An Incremental Improvement / J. Redmon, A. Farhadi // arXiv preprint arXiv:1804.02767 [Электронный ресурс]. – 2018. – 6 р. – Режим доступа: https://arxiv.org/abs/1804.02767. – Дата доступа: 04.06.2025.
28. Bochkovskiy, A. YOLOv4: optimal speed and accuracy of object detection / A. Bochkovskiy, C.-Y. Wang, H.-Y. M. Liao // arXiv preprint arXiv:2004.10934 [Электронный ресурс]. – 2020. – 17 р. – Режим доступа: https://arxiv.org/abs/2004.10934. – Дата доступа: 04.06.2025.

29. Jocher, G. YOLOv5, YOLOv8 / G. Jocher [и др.] // Ultralytics [Электронный ресурс]. – Режим доступа: https://docs.ultralytics.com. – Дата доступа: 04.06.2025.
30. Xu, Y. YOLOv6: a single-stage object detection framework for industrial appli-cations / Y. Xu, C. Zhang, Y. Wang [и др.] // Meituan [Электронный ресурс]. – 2022. – Режим доступа: https://github.com/meituan/YOLOv6. – Дата доступа: 04.06.2025.
31. Liu, S. YOLOv7: trainable bag-of-freebies sets new state-of-the-art for real-time object detectors / S. Liu, D. Wang, H. Li [и др.] // arXiv preprint arXiv:2207.02696 [Электронный ресурс]. – 2022. – 15 p. – Режим доступа: https://arxiv.org/abs/2207.02696. – Дата доступа: 04.06.2025.
32. Wang, C.-Y. YOLOv9: Learning What You Want to Learn Using Programma-ble Gradient Information / C.-Y. Wang, I.-H. Yeh, H.-Y. Mark Liao // arXiv preprint arXiv:2402.13616 [Электронный ресурс]. – 2024. – 14 p. – Режим доступа: https://arxiv.org/abs/2402.13616. – Дата доступа: 04.06.2025
33. Wang, A. YOLOv10: Real-Time End-to-End Object Detection / A. Wang, H. Yuan [и др.] // arXiv preprint arXiv:2405.14458 [Электронный ресурс]. – 2024. – 10 p. – Режим доступа: https://arxiv.org/abs/2405.14458. – Дата доступа: 04.06.2025
34. Khanam, R. YOLOv11: An Overview of the Key Architectural Enhancements / R. Khanam, M. Hussain // arXiv preprint arXiv:2410.17725 [Электронный ресурс]. – 2024. – 8 p. – Режим доступа: https://arxiv.org/abs/2410.17725. – Дата доступа: 04.06.2025
35. Tian, Y. YOLOv12: Attention-Centric Real-Time Object Detectors / Y. Tian, Q. Ye, D. Doermann // YOLOv12 Official Site [Электронный ресурс]. – 2025. – Режим доступа: https://yolov12.com. – Дата доступа: 04.06.2025
36. Pan, S. J. A survey on transfer learning / S. J. Pan, Q. Yang // IEEE Transac-tions on Knowledge and Data Engineering. – 2010. – Vol. 22(10). – P. 1345–1359.
37. Tan, C. A survey on deep transfer learning / C. Tan, F. Sun, T. Kong [и др.] // International Conference on Artificial Neural Networks (ICANN). – 2018. – P. 270–279. – DOI: 10.1007/978-3-030-01424-7_27.
38. Xia, G.-S. DOTA: a large-scale dataset for object detection in aerial images / G.-S. Xia, X. Bai, J. Ding [и др.] // CVPR. – 2018. – P. 3974–3983.
39. Zhou, P. Oriented object detection in aerial images with box attention / P. Zhou, J. Gong, Z. Wang [и др.] // arXiv preprint arXiv:1811.06805 [Электронный ресурс]. – 2018. – 10 p. – Режим доступа: https://arxiv.org/abs/1811.06805. – Дата доступа: 04.06.2025.

40. Everingham, M. The PASCAL visual object classes challenge 2010 (VOC2010) results / M. Everingham, L. Van Gool, C. Williams [и др.] // International Jour-nal of Computer Vision (IJCV). – 2010. – Vol. 88. – P. 303–338.
41. Lin, T.-Y. Microsoft COCO: common objects in context / T.-Y. Lin, M. Maire, S. Belongie [и др.] // ECCV. – 2014. – P. 740–755.
42. Rezatofighi, H. Generalized intersection over union: a metric and a loss for bounding box regression / H. Rezatofighi, N. Tsoi, J. Tan [и др.] // CVPR. – 2019. – P. 658–666.
43. Pedregosa, F. Scikit-learn: machine learning in Python / F. Pedregosa, G. Varoquaux, A. Gramfort [и др.] // Journal of Machine Learning Research. – 2011. – Vol. 12. – P. 2825–2830.
44. Ultralytics YOLO GitHub Repository [Электронный ресурс]. – Режим досту-па: https://github.com/ultralytics/ultralytics. – Дата доступа: 04.06.2025.
45. PSB Dataset // Kaggle [Электронный ресурс]. – Режим доступа: https://www.kaggle.com/datasets/yuyi1005/pcb-oriented-detection. – Дата до-ступа: 04.06.2025.
46. ShipOBB Dataset // Kaggle [Электронный ресурс]. – Режим доступа: https://www.kaggle.com/datasets/tanyajain3108/ship-detection-obb-v3. – Дата доступа: 04.06.2025.
47. DentalXRAY Dataset // Kaggle [Электронный ресурс]. – Режим доступа: https://www.kaggle.com/datasets/wanmugui/childrens-dental-panoramic-x-ray-dataset. – Дата доступа: 04.06.2025

ПРИЛОЖЕНИЕ А

ОБУЧЕНИЕ МОДЕЛЕЙ И ПОЛУЧЕНИЕ РЕЗУЛЬТАТОВ

```
REQUIRED_PACKAGES = ['ultralytics', 'matplotlib', 'pandas', 'torch']
for pkg in REQUIRED_PACKAGES:
    try:
        __import__(pkg)
    except ImportError:
        subprocess.check_call(['pip', 'install', pkg])
import os
import shutil
import subprocess
import pandas as pd
import matplotlib.pyplot as plt
import torch
from ultralytics import YOLO
from ultralytics.utils.plotting import plot_results
if not torch.cuda.is_available():
    print("CUDA не доступна. Обучение будет медленным.")
else:
    print("CUDA доступна: ускоренное обучение возможно.")

DATASETS = {
    'datasets/PSB/': 'The dataset is comprised of PCB images',
    'datasets/ShipOBB: 'Ship detection in aerial images',
    'datasets/': 'Children’s dental panoramic x-ray dataset '
}

YOLO_MODELS = [
    ('yolov8n-obb.pt', 'YOLOv8n'),
    ('yolov8s-obb.pt', 'YOLOv8s'),
    ('yolov8m-obb.pt', 'YOLOv8m'),
    ('yolov8l-obb.pt', 'YOLOv8l'),
    ('yolov8x-obb.pt', 'YOLOv8x'),
    ('yolo11n-obb.pt', 'YOLOv11n'),
    ('yolo11s-obb.pt', 'YOLOv11s'),
    ('yolo11m-obb.pt', 'YOLOv11m'),
    ('yolo11l-obb.pt', 'YOLOv11l'),
    ('yolo11x-obb.pt', 'YOLOv11x'),
```

```
    ('yolov12n-obb.pt', 'YOLOv12n'),
    ('yolov12s-obb.pt', 'YOLOv12s'),
    ('yolov12m-obb.pt', 'YOLOv12m'),
    ('yolov12l-obb.pt', 'YOLOv12l'),
    ('yolov12x-obb.pt', 'YOLOv12x')

RESULTS_DIR = "results"
os.makedirs(RESULTS_DIR, exist_ok=True)
os.makedirs("onnx_models", exist_ok=True)
os.makedirs("allModels", exist_ok=True)
summary_records = []

s.environ["PYTORCH_CUDA_ALLOC_CONF"] = "expandable_segments:True"
torch.cuda.empty_cache()
torch.cuda.ipc_collect()
gc.collect()
os.makedirs(RESULTS_DIR, exist_ok=True)
def analyze_dataset_structure(dataset_path):
    label_dir = os.path.join(dataset_path, 'labels', 'train')
    image_dir = os.path.join(dataset_path, 'images', 'train')

    label_files = [f for f in os.listdir(label_dir) if f.endswith('.txt')]
    image_files = [f for f in os.listdir(image_dir) if f.endswith(('.jpg', '.png'))]
    class_counts = {}

    for file in label_files:
        with open(os.path.join(label_dir, file)) as f:
            for line in f:
                class_id = int(line.strip().split()[0])
                class_counts[class_id] = class_counts.get(class_id, 0) + 1

    print(f"\n--- EDA for {dataset_path} ---")
    print(f"Total images: {len(image_files)}")
    print(f"Total labels: {len(label_files)}")
    print(f"Class distribution: {class_counts}")

    plt.bar(class_counts.keys(), class_counts.values())
    plt.xlabel('Class ID')
    plt.ylabel('Count')
```

```
    plt.title(f'Class distribution in {os.path.basename(dataset_path)}')

plt.savefig(f"{RESULTS_DIR}/{os.path.basename(dataset_path)}_class_dist.png"
)
    plt.close()

def train_and_evaluate(model_path, model_name, dataset_path):
    model_path_full = os.path.join("allModels", model_path)
    model = YOLO(model_path_full)
    model_name_safe = model_name.replace('/', '_')
    ds_name = os.path.basename(dataset_path)

    results = model.train(
        data=os.path.join(dataset_path, 'data.yaml'),
        epochs=30,
        imgsz=640,
        name=f"{model_name_safe}_{ds_name}",
        project=RESULTS_DIR,
        save=True,
        verbose=True
    )

    val_results = model.val()
    metrics = val_results.results_dict

    onnx_export_path = f"onnx_models/{model_name_safe}_{ds_name}.onnx"
    model.export(format='onnx', imgsz=640, dynamic=True, simplify=True,
half=False, opset=12, device='cuda' if torch.cuda.is_available() else 'cpu')

    try:
        plot_results(model,
f"{RESULTS_DIR}/{model_name_safe}_{ds_name}_metrics.png")
    except Exception as e:
        print(f"Plotting failed for {model_name}: {e}")

    return results

for dataset_path, dataset_desc in DATASETS.items():
    analyze_dataset_structure(dataset_path)
```

```
    for model_path, model_name in YOLO_MODELS:
        train_and_evaluate(model_path, model_name, dataset_path)
        torch.cuda.empty_cache()        # освобождает кеш, не память полностью
        torch.cuda.ipc_collect()        # дополнительная очистка, особенно в Colab
        gc.collect()

def train_and_evaluate(model_path, model_name, dataset_path):
    model_path_full = os.path.join("allModels", model_path)
    model = YOLO(model_path_full)
    model_name_safe = model_name.replace('/', '_')
    ds_name = os.path.basename(dataset_path)

    results = model.train(
        data=os.path.join(dataset_path, 'data.yaml'),
        epochs=30,
        imgsz=640,
        name=f"{model_name_safe}_{ds_name}",
        project=RESULTS_DIR,
        save=True,
        verbose=True
    )

    val_results = model.val()
    metrics = val_results.results_dict

    summary_records.append({
        'dataset': ds_name,
        'model': model_name,
        'mAP@0.5': metrics.get('metrics/mAP_0.5', 0),
        'mAP@0.5:0.95': metrics.get('metrics/mAP_0.5:0.95', 0),
        'precision': metrics.get('metrics/precision', 0),
        'recall': metrics.get('metrics/recall', 0),
        'IoU': metrics.get('metrics/box_iou', 0)
    })

    onnx_export_path = f"onnx_models/{model_name_safe}_{ds_name}.onnx"
    model.export(format='onnx', imgsz=640, dynamic=True, simplify=True,
half=False, opset=12, device='cuda' if torch.cuda.is_available() else 'cpu')
```

```
    try:
        plot_results(model,
f"{RESULTS_DIR}/{model_name_safe}_{ds_name}_metrics.png")
    except Exception as e:
        print(f"Plotting failed for {model_name}: {e}")

    return results

for dataset_path, dataset_desc in DATASETS.items():
    analyze_dataset_structure(dataset_path)

    for model_path, model_name in YOLO_MODELS:
        train_and_evaluate(model_path, model_name, dataset_path)

df_summary = pd.DataFrame(summary_records)
summary_path = os.path.join(RESULTS_DIR, "yolo_obb_results.csv")
df_summary.to_csv(summary_path, index=False)
print(f"\nSaved summary table: {summary_path}")

plt.figure(figsize=(12, 7))
for ds in df_summary['dataset'].unique():
    subset = df_summary[df_summary['dataset'] == ds]
    plt.plot(subset['model'], subset['mAP@0.5:0.95'], label=f"{ds}", marker='o')

plt.xlabel('YOLO Model')
plt.ylabel('mAP@0.5:0.95')
plt.title('Сравнение моделей YOLO OBB на разных датасетах')
plt.xticks(rotation=45)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig(os.path.join(RESULTS_DIR, "combined_map_comparison.png"))
plt.show()

#wandb.login()

BASE_PATHS = {
    "DentalXRAY": "/ results_DentalXRAY",
```

```
    "PSB": "/ results_PSB",
    "SHIPS": "/ results_SHIPS"
}

def init_wandb(project_name, run_name, tags):
    return wandb.init(
        project=project_name,
        name=run_name,
        config={"tags": tags},
        reinit=True
    )

def log_all_epochs(csv_path, dataset, model):
    df = pd.read_csv(csv_path)

    tags = [dataset, model]
    run_name = model


    if "YOLOv8" in model:
        model_series = "YOLOv8"
    elif "YOLOv11" in model:
        model_series = "YOLOv11"
    elif "YOLOv12" in model:
        model_series = "YOLOv12"
    else:
        model_series = "UnknownYOLO"

    general_project = f"{dataset}_ALL"
    run = init_wandb(general_project, run_name, tags)
    for i, row in df.iterrows():
        wandb.log(row.to_dict(), step=int(row["epoch"]))
    run.finish()

    model_project = f"{model_series}_{dataset}"
    run = init_wandb(model_project, run_name, tags)
    for i, row in df.iterrows():
        wandb.log(row.to_dict(), step=int(row["epoch"]))
    run.finish()
```

```
for dataset, base_path in BASE_PATHS.items():
    for subfolder in os.listdir(base_path):
        if subfolder.endswith("2"):
            continue
        folder_path = os.path.join(base_path, subfolder)
        csv_file = os.path.join(folder_path, "results.csv")
        if os.path.exists(csv_file):
            print(f"Logging: {csv_file}")
            log_all_epochs(csv_file, dataset, subfolder)
        else:
            print(f"results.csv not found in {folder_path}")
import requests

# Введите свои данные
api_key = "api-key_W&B"
entity = "entity"

headers = {"Authorization": f"Bearer {api_key}"}
url = f"https://api.wandb.ai/graphql"

query = """
query ListProjects($entity: String!) {
  entity(name: $entity) {
    projects {
      edges {
        node {
          name
        }
      }
    }
  }
}
"""

response = requests.post(url, headers=headers, json={"query": query, "variables":
{"entity": entity}})
projects = response.json()["data"]["entity"]["projects"]["edges"]
project_names = [p["node"]["name"] for p in projects]
```

```
print(f"Найдено {len(project_names)} проектов:")
for name in project_names:
    print(f" - {name}")
for name in project_names:
    print(f" Удаление проекта: {name} ...")
    delete_url = f"https://wandb.ai/entities/{entity}/projects/{name}"
    del_response = requests.delete(delete_url, headers=headers)
    if del_response.status_code == 200:
        print(" Успешно удалено")
    else:
        print(f"Ошибка удаления {name}: {del_response.status_code}")

base_dirs = {
    "DentalXRAY": "/results_DentalXRAY",
    "PSB": "/results_PSB",
    "SHIPS": "/results_SHIPS"
}

records = []

for dataset, base_path in base_dirs.items():
    for subfolder in os.listdir(base_path):
        if subfolder.endswith("2"):
            continue  # пропускаем дубликаты
        model_name = subfolder
        folder_path = os.path.join(base_path, subfolder)
        csv_file = os.path.join(folder_path, "results.csv")
        if os.path.exists(csv_file):
            df = pd.read_csv(csv_file)
            final = df.iloc[-1]  # последняя эпоха
            records.append({
                "dataset": dataset,
                "model": model_name,
                "mAP50": final["metrics/mAP50(B)"],
                "mAP50-95": final["metrics/mAP50-95(B)"],
                "precision": final["metrics/precision(B)"],
                "recall": final["metrics/recall(B)"],
                "val_box_loss": final["val/box_loss"],
```

```
            "val_cls_loss": final["val/cls_loss"],
            "val_dfl_loss": final["val/dfl_loss"]
        })
    else:
        print(f" results.csv не найден в: {folder_path}")

summary_df = pd.DataFrame.from_records(records)

summary_csv_path = "/content/drive/MyDrive/yolo_summary_table.csv"
summary_df.to_csv(summary_csv_path, index=False)

print(f" Сводная таблица сохранена в: {summary_csv_path}")
summary_df.head()
nested_tables = defaultdict(lambda: defaultdict(list))

for dataset, base_path in base_dirs.items():
    for subfolder in os.listdir(base_path):
        if subfolder.endswith("2"):
            continue
        model_name = subfolder
        folder_path = os.path.join(base_path, subfolder)
        csv_file = os.path.join(folder_path, "results.csv")

        if os.path.exists(csv_file):
            df = pd.read_csv(csv_file)
            final = df.iloc[-1]

            if "YOLOv8" in model_name:
                model_family = "YOLOv8"
            elif "YOLOv11" in model_name:
                model_family = "YOLOv11"
            elif "YOLOv12" in model_name:
                model_family = "YOLOv12"
            else:
                model_family = "Unknown"
            size = model_name.replace(model_family, "")[0]
            nested_tables[dataset][model_family].append({
                "model": model_name,
                "size": size,
```

```
            "mAP50": final["metrics/mAP50(B)"],
            "mAP50-95": final["metrics/mAP50-95(B)"],
            "precision": final["metrics/precision(B)"],
            "recall": final["metrics/recall(B)"],
            "val_box_loss": final["val/box_loss"],
            "val_cls_loss": final["val/cls_loss"],
            "val_dfl_loss": final["val/dfl_loss"]
        })
    else:
        print(f" CSV не найден: {folder_path}")
for dataset, model_families in nested_tables.items():
  for model_family, records in model_families.items():
    df = pd.DataFrame.from_records(records)
    file_path                                                    =
f"/content/drive/MyDrive/summary_{model_family}_{dataset}.csv"
    df.to_csv(file_path, index=False)
    print(f" Сохранено: {file_path}")
```

ПРИЛОЖЕНИЕ Б

СВОДНАЯ ТАБЛИЦА РЕЗУЛЬТАТОВ

dataset	model	mAP50	mAP50-95	precision	recall	box_loss	cls_loss	dfl_loss
PSB	YOLOv8n_	0,66	0,54	0,78	0,61	0,62	0,75	1,15
PSB	YOLOv8s_	0,73	0,65	0,78	0,70	0,51	0,46	1,16
PSB	YOLOv8m_	0,75	0,68	0,70	0,73	0,43	0,41	1,14
PSB	**YOLOv8l_**	**0,76**	**0,71**	**0,81**	**0,75**	**0,38**	**0,34**	**1,14**
PSB	**YOLOv8x_**	**0,84**	**0,78**	**0,90**	**0,73**	**0,38**	**0,32**	**1,15**
PSB	YOLOv11n_	0,57	0,47	0,79	0,50	0,59	0,86	1,09
PSB	YOLOv11s_	0,71	0,62	0,80	0,66	0,52	0,54	1,15
PSB	**YOLOv11m_**	**0,75**	**0,68**	**0,74**	**0,71**	**0,43**	**0,38**	**1,14**
PSB	**YOLOv11l_**	**0,75**	**0,69**	**0,73**	**0,73**	**0,43**	**0,37**	**1,16**
PSB	YOLOv12n_	0,61	0,45	0,87	0,57	0,95	0,84	0,82
PSB	**YOLOv12s_**	**0,84**	**0,69**	**0,77**	**0,83**	**0,69**	**0,37**	**0,81**
PSB	**YOLOv12m_**	**0,92**	**0,79**	**0,97**	**0,82**	**0,61**	**0,31**	**0,80**
SHIPS	YOLOv8n_	0,90	0,54	0,86	0,80	1,46	1,08	2,52
SHIPS	YOLOv8s_	0,89	0,51	0,86	0,82	1,50	1,07	2,63
SHIPS	**YOLOv8m_**	**0,83**	**0,49**	**0,83**	**0,77**	**1,52**	**1,16**	**2,81**
SHIPS	**YOLOv8l_**	**0,79**	**0,44**	**0,75**	**0,76**	**1,64**	**1,13**	**2,89**
SHIPS	YOLOv8x_	0,77	0,42	0,68	0,77	1,63	1,25	2,89
SHIPS	YOLOv11n_	0,91	0,54	0,84	0,89	1,46	1,06	2,48
SHIPS	YOLOv11s_	0,90	0,50	0,90	0,82	1,52	1,02	2,63
SHIPS	YOLOv11m_	0,88	0,51	0,82	0,83	1,48	0,98	2,80
SHIPS	**YOLOv11l_**	**0,83**	**0,46**	**0,72**	**0,81**	**1,53**	**1,07**	**2,87**
SHIPS	YOLOv12n_	0,41	0,17	0,39	0,42	1,85	1,90	2,15
SHIPS	YOLOv12s_	0,32	0,13	0,33	0,37	1,94	2,11	2,42
SHIPS	YOLOv12m_	0,28	0,11	0,32	0,31	2,01	2,23	2,52
DentalXRAY	YOLOv8n_	0,78	0,62	0,69	0,80	0,85	1,20	1,45
DentalXRAY	YOLOv8s_	0,87	0,73	0,76	0,82	0,74	0,86	1,43
DentalXRAY	**YOLOv8m_**	**0,96**	**0,83**	**0,85**	**0,89**	**0,64**	**0,70**	**1,36**
DentalXRAY	**YOLOv8l_**	**0,97**	**0,85**	**0,90**	**0,90**	**0,58**	**0,63**	**1,35**
DentalXRAY	YOLOv11n_	0,72	0,55	0,63	0,65	0,88	1,24	1,43
DentalXRAY	YOLOv11s_	0,86	0,72	0,86	0,72	0,75	0,93	1,40
DentalXRAY	**YOLOv11m_**	**0,93**	**0,80**	**0,83**	**0,89**	**0,68**	**0,71**	**1,39**
DentalXRAY	**YOLOv11l_**	**0,89**	**0,76**	**0,83**	**0,90**	**0,67**	**0,70**	**1,38**
DentalXRAY	YOLOv12n_	0,86	0,53	0,82	0,76	1,36	1,24	1,12
DentalXRAY	YOLOv12s_	0,92	0,58	0,84	0,87	1,31	0,97	1,13
DentalXRAY	YOLOv12m_	0,95	0,63	0,93	0,87	1,29	0,92	1,10
DentalXRAY	YOLOv12l_	0,91	0,58	0,83	0,87	1,29	0,97	1,10