МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ»
(национальный исследовательский университет)»

Институт (Филиал) № 8 «Компьютерные науки и прикладная математика» **Кафедра** 806

Группа М8О-401Б-21 **Направление подготовки** 01.03.02 «Прикладная математика и информатика»

Профиль Информатика

Квалификация: бакалавр

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

на тему: Разработка экстрактора экспериментальных данных из графического представления

Автор ВКРБ: Тулин Иван Денисович (_____)

Руководитель: Морозов Александр Юрьевич (_____)

Консультант: — (_____)

Консультант: — (_____)

Рецензент: — (_____)

К защите допустить

Заведующий кафедрой № 806 Крылов Сергей Сергеевич (_____)

____ мая 2025 года

Москва 2025

РЕФЕРАТ

Выпускная квалификационная работа бакалавра состоит из 47 страниц, 4 рисунков, 1 таблицы, 14 использованных источников.

А ВТОМАТИЧЕСКОЕ ИЗВЛЕЧЕНИЕ ГРАФИЧЕСКИХ ДАННЫХ, ГРАФИЧЕСКИЕ ПРЕДСТАВЛЕНИЯ, КОМПЬЮТЕРНОЕ ЗРЕНИЕ, ГЛУБОКОЕ ОБУЧЕНИЕ, ДЕТЕКЦИЯ ОБЪЕКТОВ, СИНТЕТИЧЕСКИЕ ДАТАСЕТЫ.

Настоящая работа посвящена разработке и исследованию методов автоматического извлечения числовых данных из двумерных графических представлений (линейных графиков, диаграмм рассеяния, кривых) с использованием классических алгоритмов компьютерного зрения и современных архитектур глубокого обучения. Теоретическая часть охватывает этапы предобработки (фильтрация, бинаризация), детекции областей интереса (линии сетки, оси), сегментации графических элементов и обратного преобразования пиксельных координат в реальные значения. Практическая часть описывает процесс генерации синтетического датасета с автоматической разметкой, построение end-to-end пайплайна на основе YOLOv8 (детекция области, детекция точек) и метода наименьших квадратов для аппроксимации, а также оценку моделей по метрикам IoU, Precision/Recall, MSE и RMSE. В заключении приведены рекомендации по расширению домена данных, добавлению OCR-модуля и поддержке instance-segmentation.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ 6
ВВЕДЕНИЕ 7
1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 10
1.1 Графическое представление числовых данных 10
1.1.1 Классификация графиков 10
1.1.2 Координатные системы и масштабы 11
1.1.3 Вывод 12
1.2 Постановка задачи 13
1.2.1 Исходные предположения 13
1.2.2 Вывод 14
1.3 Основы компьютерного зрения для работы с графиками 14
1.3.1 Растровая геометрия. 15
1.3.2 Предобработка изображения 15
1.3.3 Пост-обработка примитивов 16
1.3.4 Вывод 16
1.4 Введение в архитектуру моделей детекции 16
1.4.1 Архитектура современных детекторов 17
1.4.2 Bounding Box и метрика IoU 17
1.4.3 Fast R-CNN 18
1.4.4 Faster R-CNN 19
1.4.5 Mask R-CNN 20
1.4.6 SSD 21
1.4.7 YOLO 23
1.4.8 Метрики оценки детекторов 25
1.4.9 Вывод 27
1.5 Математические методы восстановления функции 27
1.5.1 Метод наименьших квадратов (МНК) 27
1.5.2 Выбор модели аппроксимации 28
1.5.3 Оценка погрешности 29
1.5.4 Вывод 30
2 ПРАКТИЧЕСКАЯ ЧАСТЬ 31
2.1 Среда разработки и инструментарий 31
2.1.1 Аппаратная конфигурация 31

2.1.2 Программное обеспечение 31
2.2 Генерация и разметка данных 32
2.2.1 Описание существующего генератора синтетических данных 32
2.2.2 Структура классов генератора 32
2.2.3 Формат выходных данных генератора 33
2.2.4 Формирование наборов данных для обучения и тестирования 33
2.2.5 Стратегии аугментации данных 33
2.3 Модуль 1: Детекция области графика 34
2.3.1 Подготовка датасета для задачи детекции графиков (GraphDet) 34
2.3.2 Обучение нейросетевой модели YOLOv8 34
2.3.3 Метрики оценки качества детекции области графиков . . . 35
2.3.4 Пост-обработка результатов и извлечение области графика 35
2.4 Модуль 2: Детекция точек внутри графика 35
2.4.1 Подготовка датасета для детекции точек (PointDet) 35
2.4.2 Обучение нейросети YOLO для задачи детекции точек . . 36
2.4.3 Метрики качества детекции точек 37
2.4.4 Пост-обработка результатов детекции точек 37
2.5 Модуль 3: Восстановление функции методом наименьших квадратов 37
2.5.1 Обратное преобразование координат точек 38
2.5.2 Выбор математической модели аппроксимации 38
2.5.3 Реализация метода наименьших квадратов 38
2.5.4 Оценка качества аппроксимации функции 39
2.6 Сборка end-to-end пайплайна автоматического извлечения данных из графиков 39
2.6.1 Архитектура и компоненты системы 39
2.7 Экспериментальные исследования системы 40
2.7.1 Базовый сценарий (без шума) 40
2.7.2 Влияние загрязняющих факторов 41
2.7.3 Сравнение размеров моделей YOLO 42
2.7.4 Выбор степени полинома 43
2.7.5 Сводные результаты 44
3 ЗАКЛЮЧЕНИЕ 45
3.1 Достижения и сильные стороны решения 45

3.2 Ограничения решения 45
3.3 Рекомендации по дальнейшему развитию 46
3.4 Общий итог 46
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 47

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей выпускной квалификационной работе бакалавра применяют следующие сокращения и обозначения:

CNN — convolutional neural network

SSD — single-shot multibox detector

YOLO — YOLO

IoU — intersection over union

mAP — mean average precision

NMS — mean average precision

BCE — binary cross-entropy

МНК — метод наименьших квадратов

HE — histogram equalization

CLAHE — contrast limited adaptive histogram equalization

MSE — среднеквадратичная ошибка

Corr — коэффициент корреляции

ЯП — язык программирования

ВВЕДЕНИЕ

В современной научной и аналитической практике графическое представление данных стало общепринятым стандартом визуализации информации. Особенно это характерно для публикаций в научных журналах, экономических отчетов и технической документации. Однако зачастую авторы предоставляют лишь конечные графические результаты, не сопровождая их исходными числовыми данными. Причины этого могут быть различны: от желания сохранить конфиденциальность информации до банального отсутствия места для публикации полного массива данных. Такая ситуация создает существенные трудности для исследователей, которым требуется работать с этими данными. Возникает необходимость либо в повторном проведении экспериментов (что часто невозможно), либо в ручном извлечении числовых значений непосредственно из графического представления. Последний подход, хотя и кажется на первый взгляд простым (ведь человеческий глаз достаточно точно определяет положение точек на графике), на практике оказывается крайне трудоемким и подверженным ошибкам процессом. Особенно это касается случаев, когда требуется обработать большое количество графиков или когда точность извлечения данных критически важна для последующего анализа. Иными словами, для решения задачи оцифровки графиков необходима такая система, которая сводила бы к минимуму ручную работу и позволяла бы пользователю быстро и точно преобразовывать визуальные элементы в набор числовых координат. Существует два принципиально разных подхода к её реализации. Полуавтоматические инструменты, представленные, например, программами GetData Graph Digitizer и WebPlotDigitizer, предусматривают начальную ручную калибровку: пользователь отмечает на изображении контрольные точки на осях или выделяет области интереса, а затем система с помощью цветовой фильтрации в пространстве RGB или HSV, морфологических операций автоматически извлекает линии, столбцы или маркеры. Такие решения отличаются сравнительно небольшими аппаратными требованиями и поддерживают широкий спектр типов графиков — от линейных диаграмм и гистограмм до scatter-плотов и полярных представлений — однако без регулярного вмешательства пользователя для уточнения калибровки и проверки полученных точек их точность может существенно падать, особенно

на фоне сложного оформления или при плотной компоновке элементов. В полностью автоматических системах, основанных на современных подходах машинного обучения, таких как Scatteract и ChartOCR, весь процесс «от пикселя до числа» выстроен в единую конвейерную схему: например, Scatteract сочетает Faster R-CNN для детекции маркеров, Tesseract OCR для распознавания подписей осей и алгоритм RANSAC для вычисления аффинного преобразования, но при этом ограничивается только scatter-плотами и требует строго стандартизированных стилей. ChartOCR объединяет U-Net-подобную сегментацию компонентов (оси, линии, текстовые области) с последующим OCR и алгоритмами сопоставления пикселей и числовых меток, что позволяет обрабатывать разнообразные типы диаграмм, но предъявляет высокие требования к вычислительным ресурсам и чаще всего требует наличия GPU для эффективной работы. Итак, основная цель данной выпускной квалификационной работы заключается в разработке программной системы, предназначенной для автоматического извлечения числовых данных из графических изображений. Разрабатываемая система должна принимать на вход цветное двумерное изображение графика размером $m \times n \times 3$, где m — ширина изображения, n — высота, а 3 соответствует числу цветовых каналов (RGB). Результатом работы системы должна стать упорядоченная последовательность координатных пар $\{x_i, y_i\}$, где $i = 1, 2, ..., k$ а k — количество точек, составляющих кривую или графическое представление исходной функции. Научная новизна предлагаемого подхода состоит в стремлении объединить высокий уровень автоматизации процесса с минимальными требованиями к вычислительным ресурсам. В отличие от существующих систем, базирующихся на тяжёлых архитектурах глубокого обучения, данная работа предполагает использование компактных моделей и методов, позволяющих обеспечить полноценную обработку данных даже в условиях ограниченных аппаратных возможностей. Для достижения поставленной цели необходимо поэтапно решить ряд задач, каждая из которых относится к определённой области анализа визуальной информации. Во-первых, необходимо разработать метод обнаружения области, содержащей графическое представление, на общем изображении. Во-вторых, требуется классифицировать тип графика, поскольку от этого будет зависеть выбор алгоритма последующей обработки. Третьим шагом является выявление и локализация ключевых точек, формирующих кривую или другие элементы

графика, содержащие числовую информацию. И, наконец, необходимо восстановить аппроксимирующую функцию или непрерывное представление на основе дискретного набора извлечённых точек, используя подходящие методы интерполяции или регрессии. Таким образом, в рамках данной работы объектом исследования выступают методы анализа и обработки графических данных, а предметом — конкретные алгоритмы обнаружения и классификации графических элементов, а также восстановления числовых значений, закодированных в визуальной форме. В качестве рабочей гипотезы предполагается, что при использовании комбинации малозатратных по ресурсам моделей машинного обучения (например, лёгких сверточных детекторов) в сочетании с методами функциональной интерполяции возможно создать полностью автоматизированную систему извлечения данных из изображений графиков, обеспечивающую приемлемую точность при минимальной нагрузке на вычислительную среду.

1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1 Графическое представление числовых данных

Визуализация данных — это не «украшение» отчёта, а самостоятельный этап аналитического процесса, выполняющий сразу несколько ключевых функций:

- **формирование гипотез** — человек замечает тренды, кластеры и выбросы быстрее на картинке, чем в таблице;
- **коммуникация результатов** — наглядная диаграмма снижает когнитивную нагрузку на аудиторию: демонстрируется не только факт, но и контекст — масштаб изменения, сравнительное положение, доверительный интервал;
- **контроль качества данных** — графики мгновенно выявляют монтажные и сенсорные ошибки, а также нарушения предпосылок модели (нелинейность, неоднородность дисперсии);
- **сокрытие/раскрытие сложности** — хорошая визуализация позволяет «просверлить» данные до нужного уровня детализации, сохраняя при этом целостность повествования.

1.1.1 Классификация графиков

Существует множество способов классифицировать диаграммы: по типу отображаемых данных (количественные, категориальные, временные), по числу отображаемых переменных, по типу используемого визуального канала (длина, угол, цвет, насыщенность). Ниже приведена классификация, ориентированная на задачу автоматического извлечения данных:

- **линейный график** — непрерывная линия, отображающая зависимость $y = f(x)$; возможны ступенчатый, area-и multi-series варианты;
- **диаграмма рассеяния (scatter plot)** — отдельные маркеры для каждой выборки, часто с категориальной окраской;
- **столбчатая диаграмма (bar chart)** — прямоугольники равной ширины, высота которых пропорциональна величине показателя;
- **гистограмма и полигон распределения** — смежные прямоугольники, характеризующие частоты; полигон соединяет

вершины столбцов ломаной;

- **диаграммы «ящик с усами» и violin-plot** — компактное представление распределения по квинтилям и плотности;
- **тепловые карты и поверхностные визуализации (heatmap / surface)** — матрица ячеек или сеточная поверхность, в которых цвет кодирует числовое значение;
- **комбинированные и многоосевые графики** — совмещение разных типов (линия + столбцы, двойные оси) в одном кадре;
- **малые мультипликативы (small multiples)** — решётка однотипных подграфиков, каждый из которых иллюстрирует поднабор данных.

Несмотря на разнообразие форм, все перечисленные графики служат одной цели — отображению зависимости между переменными. Если говорить об автоматическом извлечении данных из графиков, описанное разнообразие способов их визуализации приводят к необходимости корректного распознавания базовых графических примитивов (линий, маркеров, прямоугольников, ячеек) и учитывания контекста: числа осей, типов шкал (линейные шкалы или логарифмические), наличия скрытых категорий. Определив класс визуализации, можно выбирать правила нормализации координат, ожидаемое число точек и метод последующей аппроксимации.

1.1.2 Координатные системы и масштабы

Любой график реализует отображение

$$(x, y) \longmapsto (u, v) \in \mathbb{Z}^2,$$

где (x, y) — числовые данные, (u, v) — координаты пикселя на растровой сетке.

Преобразование задаётся двумя компонентами:

- геометрией, то есть выбором системы координат,
- шкалированием (scale), определяющим линейность либо нелинейность осей.

Чаще всего встречаются следующие системы координат:

- **декартова (линейная) система** — большинство прикладных графиков строится в прямоугольной системе с равномерным масштабом
  - преимущества — интуитивная интерпретация: одинаковые отрезки на оси соответствуют одинаковым

приращениям величины,

– недостатки — неэффективно отображаются экспоненциальные процессы; малые значения «слипаются» у нуля;

Формально линейное шкалирование описывается аффинным преобразованием

$$u = \alpha_x\, x + \beta_x, \qquad v = \alpha_y\, y + \beta_y, \tag{1}$$

где коэффициенты $\alpha_{x,y}, \beta_{x,y}$ определяются размером ограничивающего прямоугольника $B = (x_{\min}, x_{\max}, y_{\min}, y_{\max})$.

– **логарифмическая шкала** — логарифмическая ось применяется, когда данные охватывают несколько порядков или подчиняются степенным законам. Преобразование уже не является аффинным:

$$u = \alpha_x \log_{b_x} x + \beta_x, \qquad v = \alpha_y \log_{b_y} y + \beta_y, \tag{2}$$

где b_x, b_y — основания логарифмов; для автоматического извлечения критично корректно определить логарифмический масштаб: прямая линия в лог-лог координатах описывает степенную зависимость $y \sim x^k$; если ошибочно трактовать её как линейную, метод восстановления функции даст искажённый результат;

– **полярные и географические координаты** — извлечение данных из полярных графиков (радиус-угол) или картографических проекций требует дополнительного этапа — преобразования к декартовым координатам во время пост-обработки (хотя в данной работе акцент сделан на декартовой системе).

1.1.3 Вывод

Графическое представление числовых данных — многослойная концепция, включающая выбор визуального примитива, системы координат и масштаба осей. От корректности учёта каждой из этих составляющих зависит успех последующего этапа экстрагирования: алгоритм должен «понимать», что линия на логарифмическом графике кодирует $\log y$, а не y, и что одна точка может относиться к сложной мультисерийной структуре.

1.2 Постановка задачи

На вход подаётся цветное растровое изображение

$$I \in [0,1]^{m \times n \times 3}, \tag{3}$$

где $m \times n$ — размеры картинки, содержащее K двумерных линейных графиков. Пусть исходные дискретные данные, по которым построены графики, заданы множеством

$$D = \left\{ \{(x_i^j,\ y_i^j)\}_{i=1}^{N_j} \right\}_{j=1}^{K},$$

где x_i^j и y_i^j — значения независимой и зависимой переменной соответственно, а N_j — число точек j-го графика. Пользователь U интерполировал каждую выборку, получив непрерывную аппроксимацию $\hat{f}_j(x)$, и вывел все K кривых на одном изображении I.

1.2.1 Исходные предположения

В дальнейшей работе воспользуемся следующими предположениями

- **линейная шкала** — оси графиков имеют линейный масштаб, при нелинейной шкале используется известное преобразование, например при логарифмической оси x: $\hat{x} = x_{\min}\left(\frac{x_{\max}}{x_{\min}}\right)^{x}$;
- **ограничивающий прямоугольник** — все кривые лежат внутри рамки $B = (x_{\min},\, x_{\max},\, y_{\min},\, y_{\max})$, где $x_{\min} = \min x_i^j,\ x_{\max} = \max x_i^j$ (аналогично для y); автоматически выделить B сложно, поэтому координаты задаёт оператор;
- **нормализация** — после обрезки рамки все кривые приводятся к единичному квадрату $B_S = (0,1,0,1)$; для денормализации используется

$$\hat{x} = x\, x_{\max} + (1-x)\, x_{\min},$$

$$\hat{y} = y\, y_{\max} + (1-y)\, y_{\min};$$

- **фиксированная сетка по** x — для каждого графика предполагается одинаковое число отсчётов N; узлы равномерны:

$$X = (x_1, \ldots, x_N), \qquad x_i = \frac{i-1}{N-1};$$

сеть предсказывает только ординаты

$$Y = (y_1, \ldots, y_N)$$

.

Необходимо построить метод, который по изображению I восстанавливает набор нормализованных пар

$$\tilde{D} = \left\{ \{(x_i,\ \tilde{y}_i^j)\}_{i=1}^{N} \right\}_{j=1}^{K},$$

где x_i фиксируются сеткой, а $\tilde{y}_i^j$ максимально приближаются к истинным y_i^j. После денормализации и интерполяции кривые $\hat{f}_j^{(\text{pred})}(x)$ должны как можно точнее совпадать с исходными $\hat{f}_j(x)$.

1.2.2 Вывод

Сформулированный подход гарантирует воспроизводимость эксперимента, позволяет количественно оценивать погрешность восстановления и создаёт основу для дальнейшего конструирования алгоритма — от выбора архитектуры нейросети до тонкой настройки процедуры обучения и валидации.

1.3 Основы компьютерного зрения для работы с графиками

Цифровое изображение — это дискретная двумерная функция

$$I \colon \mathbb{Z}^2 \to \mathbb{R}^C,$$

где каждой паре целочисленных координат (u, v) сопоставляется C-компонентный вектор интенсивностей (при стандартном 24-битном RGB, $C = 3$, каждый канал кодируется 8 битами).

- **RGB.** Наиболее распространённое пространство, где компоненты $(R, G, B) \in [0, 255]^3$ описывают аддитивное смешение цветов. Преимущество — простота; недостаток — корреляция каналов, препятствующая пороговой сегментации тонких линий.
- **HSV/HSI.** Декоррелирует цветовой тон H, насыщенность S и яркость V/I. Удобно отделять цветные кривые графиков от однотонной подложки (сетка, подписи).
- **YCbCr.** Применяется в видео-кодеках; яркостная компонента Y отделена от цветоразностных Cb, Cr, что облегчает бинаризацию.

1.3.1 Растровая геометрия.

Каждый пиксель занимает единичную ячейку регулярной решётки. Координаты (u, v) интерпретируются как центры ячеек при построении математической модели Для линейных графиков критично соотносить ширину линии (часто 1 px) с размером шага Δu сетки, чтобы избежать потери информации при субдискретных преобразованиях (масштабирование, поворот).

1.3.2 Предобработка изображения

Предобработка уменьшает влияние шума, повышает контраст и подготавливает изображение к поиску примитивов. Предобработка бывает следующих видов:

- **Фильтрация.** Цель фильтрации — подавить шум при сохранении тонких линий графика.
  - Гауссов фильтр. Свертка с изотропным ядром подавляет высокочастотные помехи. Значение σ выбирают из условия $\sigma \leq 1{,}0$ px, чтобы не размыть тонкие линии.
  - Медианная фильтрация. Эффективна против импульсного шума, сохраняет резкие границы линий.
  - Морфологические операции. Утолщение dilation и эрозия erosion применяются для разрыва сплошных сеток или заполнения пробелов в контуре линий.
- **Повышение контраста.**
  - Выравнивание гистограммы (HE).[1] Перераспределяет уровни яркости, улучшая различимость линии и фона.
  - CLAHE. Адаптивный вариант HE с ограничением контраста, предотвращающий пересвет тонких объектов на ярком фоне.
- **Пороговая бинаризация.**
  - Глобальный порог (метод Отсу)[2] — подходит для чётких чёрно-белых графиков без сетки.
  - Адаптивный порог — вычисляется по локальному окну, надёжен при неравномерном освещении или градиентной заливке фона.

Полученная бинарная маска $B(u, v) \in \{0, 1\}$ служит входом для выделения геометрических примитивов.

1.3.3 Пост-обработка примитивов

После выделения линия/точка переводится из растровых координат (u, v) в нормализованные (x, y) по формулам (1), (2). Кластеризация помогает раздельно сохранить несколько цветовых серий, если линии интерферируют.

1.3.4 Вывод

Представленный инструментарий компьютерного зрения обеспечивает устойчивое выделение базовых элементов графика (оси, сетка, кривая, маркеры), минимизируя влияние шума и неоднородного освещения.

1.4 Введение в архитектуру моделей детекции

Задача детекции — это одновременное определение локализации и класса объектов на изображении, видео-потоке или иной среде. Формально модель выдаёт для каждого объекта прямоугольник с координатами и принадлежность к одному из K классов. В отличие от классификации, где прогнозируется ровно один класс для всего кадра, здесь допускается неопределённое число объектов. Задачи детекции бывают следующих направленностей:

- **детекция объектов (Object Detection)** — локализация всех объектов из конечного набора классов;
- **детекция аномалий** — поиск неизвестных/дефектных областей
- **детекция аномалий** — выявление отличия между парами изображений во времени
- **key-point / pose detection** — очки скелета, опорные точки лица

В настоящей работе мы будем рассматривать задачу детекции объектов с применением нейросетевых моделей. Нейросетевые модели детекции делятся на два вида:

- **двухстадийные** — первой стадией уточняется набор областей, где возможно есть объекты (применяется Selective Search, либо RPN), а второй стадией происходит уточнение координат BBox'а и классификация объектов; примеры двухстадийных детекторов:

    - Fast R-CNN,
    - Faster R-CNN,
    - Mask R-CNN;
- **одностадийные** — детекция выполняется одной стадией — ко входному изображению применяется несколько конволюционных слоев для получения feature map, затем накладываются default box'ы (или anchor'ы) и из них уже происходит регрессия координат объектов и дальнейшая классификация; примеры одностадийных детекторов:
    - YOLO,
    - SSD.

1.4.1 Архитектура современных детекторов

Одностадийные против двухстадийных детекторов — компромисс «скорость/точность» смещается в пользу one-stage благодаря RetinaNet, EfficientDet и новым трансформерам. **Backbone** — экстракция признаков (ResNet, CSP-Darknet, Swin-T). **Neck** — слияние признаков разных масштабов (FPN, PANet, Bi-FPN). **Head** — прогноз (class, bbox, при необходимости mask/pose).

1.4.2 Bounding Box и метрика IoU

Bounding box — минимальный прямоугольник, охватывающий интересующий объект на изображении. Для классических (axis-aligned) боксов используют два эквивалентных представления:

- углы: $(x_{\min},\ y_{\min},\ x_{\max},\ y_{\max})$;
- центр + габариты: $(x_c,\ y_c,\ w,\ h)$.

BBox служит «упрощённой геометрией» объекта: модели достаточно координат прямоугольника, чтобы указать, где находится объект и какую часть кадра считать «его». Он остаётся стандартом для задач, где важны простота и скорость (детекция, трекинг, визуальный поиск и др.).

Метрика IoU оценивает, насколько две области (обычно предсказанный и истинный bbox) совпадают:

$$\text{IoU}(A, B) = \frac{|A \cap B|}{|A \cup B|}, \qquad 0 \leq \text{IoU} \leq 1. \tag{4}$$

$|A \cap B|$ — площадь пересечения, $|A \cup B|$ — площадь объединения.

- IoU $= 1$ — идеальное совпадение.
- IoU $= 0$ — области не пересекаются.

Применения:

- В тренировке/валидации определяет, считать ли детекцию True Positive (например, IoU ≥ 0.5 в VOC или ≥ 0.75 в COCO при «stricter» подсчёте).
- В алгоритмах якорей и NMS пороги IoU регулируют, какие боксы объединять или отбрасывать.
- Чувствительность IoU к геометрической точности делает её ключевой при вычислении AP/mAP: ошибка в несколько пикселей заметно снижает метрику на высоких (0.9+) порогах.

Таким образом, bbox компактно описывает положение объекта, а IoU даёт строгую числовую оценку совпадения предсказания с истиной.

1.4.3 Fast R-CNN

Fast R-CNN (Fast Region-based Convolutional Network) — это двухстадийный детектор объектов, предложенный Россом Гиршиком в 2015 году.

Пайплайн инференса

1) **вход** — RGB-кадр произвольного размера (обычно масштабируется так, чтобы длинная сторона ≈ 1000 px, короткая ≤ 600 px);
2) **общие свёртки (Backbone)** — конволюционная сеть обрабатывает изображение один раз, формируя трёхмерную карту признаков;
3) **Region Proposals** — внешний алгоритм Selective Search/EdgeBoxes выдаёт около 2 000 кандидатов;
4) **RoI Pooling** — каждый регион проецируется в координаты карты и разбивается на ячейки; из каждой берётся максимум, формируя тензор фиксированного размера;
5) **полносвязные слои (FC6, FC7)** — преобразуют тензор в вектор признаков;
6) **две «сестринские» головы:**
   - **Softmax** $\rightarrow$ распределение p_i по $K+1$ классам («фон» $+K$ объектов);
   - **BBox-регрессор** $\rightarrow$ вектор смещений Δt_i ($4K$ либо

4 координаты) для уточнения границ окна.

7) **Non Maximum Suppression (NMS)** удаляет перекрывающиеся окна.

Многозадачный loss

$$\mathcal{L}(p, u, t^*, t) = \mathcal{L}_{\text{cls}}(p, u) + \lambda[u \geq 1]\mathcal{L}_{\text{loc}}(t^*, t), \tag{5}$$

где u — истинный класс (0 — фон), $\mathcal{L}_{\text{cls}}$ — log-loss, $\mathcal{L}_{\text{loc}}$ — Smooth L1 между предсказанными и таргет-парами бокса, $\lambda = 1$. В регрессии участвуют лишь положительные RoI.

Наследие и влияние Fast R-CNN стал фундаментом для:

– **Faster R-CNN** — встроенный RPN заменил внешние предложения,
– **Mask R-CNN** — RoIAlign и параллельная сегментация,
– Десятков модификаций (OHEM, C-IoU loss, attention-RoI и др.).

1.4.4 Faster R-CNN

Несмотря на ускорение Fast R-CNN, эвристический алгоритм Selective Search требовал до $2\,\text{s}$ CPU на кадр. Faster R-CNN вводит Region Proposal Network (RPN), позволяя формировать окна за $< 10\,\text{ms}$ прямо на GPU и превращая весь конвейер в единую, полностью обучаемую CNN.

Компоненты модели:

а) Backbone. Сверточная сеть создаёт карту признаков с шагом 16 px после уменьшения изображения (длинная сторона $\approx 1000\,\text{px}$, короткая $\approx 600\,\text{px}$).

б) Region Proposal Network (RPN).
   – На каждой клетке карты скользит свёртка 3×3.
   – Два параллельных слоя 1×1 предсказывают: объектность (логиты «object / background»); четыре смещения бокса.
   – Для охвата разных форм используются anchors: 9 прямоугольников (масштабы $128^2, 256^2, 512^2$ px $\times$ соотношения 1:1, 1:2, 2:1) привязаны к центру каждой клетки.

в) Отбор предложений. RPN сортирует якоря по вероятности «object», затем применяет NMS и оставляет ≈ 300 окон.

г) RoI Pooling + детектор Fast R-CNN. Каждое окно проецируют в координаты фич-карты и разбивают на сетку 7×7; max-pooling формирует тензор фиксированного размера, который после двух

FC-слоёв расходится на softmax по $K+1$ классам и box-регрессор.

д) Финальный NMS. Удаляет пересекающиеся детекции, формируя итоговый набор боксов с метками и вероятностями.

Обучение. Используются два одновременных лосса с общими свёрточными весами.

Влияние. Faster R-CNN стал основой для каскадных и гибридных детекторов (Cascade R-CNN, Libra, FCOS-TSP, RT-DETR) и остаётся надёжной отправной точкой для исследований, где точность важнее латентности.

1.4.5 Mask R-CNN

Назначение. Mask R-CNN расширяет Faster R-CNN, добавляя полноценную ветку семантической сегментации, при этом сохраняет двухступенчатую логику детекции. Модель одновременно решает: (i) где находится объект (bbox) и (ii) какие пиксели ему принадлежат (mask). Дополнительная задача замедляет инференс лишь на $\approx 10\,\%$; на GPU K40 достигается ~ 5 fps, так как новая функциональность использует уже вычисленные признаки.

Сквозной поток данных.

а) Backbone + Feature Pyramid Network (FPN). В качестве backbone применяется ResNet-50/101, «обёрнутый» FPN. Последняя формирует многоуровневую пирамиду признаков $\{P_2, \ldots, P_6\}$, обеспечивая одинаково высокое качество на объектах разных масштабов.

б) Region Proposal Network. Тот же RPN, что и в Faster R-CNN, скользит свёрткой 3×3 по самой детальной карте пирамиды, предсказывая «object / background» и смещения anchor-боксов. 300 кандидатов с максимальной объектностью передаются далее.

в) RoIAlign — ключевое нововведение. В Fast/Faster RoI Pooling вносил ошибку округления. RoIAlign устраняет её:

- координаты окна не округляются при проекции на карту признаков;
- внутри pooled-ячеек значения берутся точной билинейной интерполяцией.

Это дало до $+50\,\%$ относительного прироста точности масок на

строгих IoU-порогах.

г) Две параллельные головы.

- **Detection-head** повторяет Fast R-CNN: два FC слоя $\rightarrow$ (a) softmax по $K+1$ классам, (b) бокс-регрессор.
- **Mask-head** — небольшой FCN: четыре свёртки 3×3 (256 карт признаков), затем транпон. свёртка (stride 2) до разрешения 28×28. Для каждого RoI формируется тензор K каналов; лосс вычисляется лишь по каналу истинного класса.

д) Сбор результатов. После второго NMS каждый объект получает bbox, категорию, confidence и бинарную маску; маска поднимается (bilinear upsample) до исходного размера бокса и обрезается по его границам.

Многозадачный лосс.

$$\mathcal{L} = \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{box}} + \mathcal{L}_{\text{mask}}, \tag{6}$$

- $\mathcal{L}_{\text{cls}}$ — softmax-log-loss на $K+1$ классах;
- $\mathcal{L}_{\text{box}}$ — Smooth L1 для координат бокса;
- $\mathcal{L}_{\text{mask}}$ — бинарная cross-entropy на карте $28{\times}28$, считаемая только для положительных RoI.

Все компоненты взвешены одинаково ($\lambda = 1$), что исключает подбор гиперпараметров.

Итог. Благодаря RoIAlign и FPN Mask R-CNN поднял state-of-the-art по instance-segmentation, оставаясь практически настолько же быстрым, как Faster R-CNN.

1.4.6 SSD

Детекторы линейки R-CNN достигают высокой точности, но любая «предварительная» стадия (Selective Search, RPN) увеличивает латентность и усложняет код. SSD полностью отказывается от предложений областей: сеть единственным проходом предсказывает классы и смещения для фиксированного набора default boxes, распределённых по нескольким уровням карты признаков. Такая схема обеспечила 40–60 fps на Titan X при точности, сопоставимой с Faster R-CNN.

Внутреннее устройство.

а) Базовая сеть и псевдо-пирамида. В SSD-300 берут VGG-16 до слоя

conv5_3, затем добавляют пять сверточных блоков conv8_2 – conv11_2, формируя шесть карт масштабов 38×38, 19×19, 10×10, 5×5, 3×3, 1×1. Это «пирамида» без top-down-слияния, которое позже обеспечит FPN.

б) Default boxes. Для каждой ячейки каждой карты задают набор прямоугольников с парами «масштаб + аспект-ratio»:
   - Масштаб s_k — линейная прогрессия от 0.1 до 0.9 (у SSD-512 — 0.07…0.93).
   - Аспекты — $\{1, 2, 3, 1/2, 1/3\}$.

   В SSD-300 формируется $\approx$ 8000 коробок. Для каждой сеть предсказывает $(K+1)$ логитов класса и четыре оффсета $(\Delta x, \Delta y, \Delta w, \Delta h)$. Отдельный логит objectness не нужен — он входит в softmax.

в) MultiBox-лосс. Суммирует

$$\mathcal{L} = \mathcal{L}_{\text{conf}} + \mathcal{L}_{\text{loc}}, \tag{7}$$

   где $\mathcal{L}_{\text{conf}}$ — softmax-loss по классам для всех default boxes, а $\mathcal{L}_{\text{loc}}$ — Smooth L1, считанная только для коробок с IoU ≥ 0.5 с любым GT. Чтобы негативы не ”забили”позитивы, применяют *hard negative mining*: выбирают не более трёх самых «уверенных, но ошибочных» отрицательных примеров на один положительный.

г) NMS на выходе. После softmax для каждой категории берут до 10 наиболее уверенных коробок и применяют NMS, оставляя до 200 итоговых детекций.

Трюки, повысившие скорость и точность.

– Random expansion. С вероятностью 0.5 изображение обрамляют белой рамкой до 300 %, сохраняя мелкие объекты.
– Photometric distortions. Случайные hue/saturation и перестановка каналов помогают контрастной инвариантности.
– Разрешение входа. SSD-300 (300×300) — максимум FPS; SSD-512 (512×512) даёт +3–5 pp mAP за счёт мелких объектов, но снижает FPS вдвое.
– Atrous и depthwise свёртки. Привели к SSD-Lite: MobileNetV2 + depthwise свёртки дают 20 fps на мобильном GPU при энергопотреблении < 1 W.

Ограничения и эволюция. Главная слабость — мелкие объекты. Улучшения:

- **DSSD** — добавил де-конволюционный путь, +2 pp mAP, но 9 fps.
- **FPN-SSD, MDSSD, Tiny-SSD** — объединяют идеи пирамиды признаков и лёгких бекбонов.
- **SSD-Lite** — depthwise свёртки + MobileNetV2 для realtime на ARM SoC.

Итог. SSD стал первым массовым детектором «без предложений»: идея мульти-скейл-прогноза из нескольких карт лежит в основе современных real-time моделей (YOLOv8/v9, EfficientDet-D, RT-DETR-lite). Для задач, где критична скорость на CPU или мобильном GPU, SSD-Lite остаётся надёжной отправной точкой.

1.4.7 YOLO

YOLO You Only Look Once [3] превращает поиск объектов в задачу единой регрессии: кадр пропускается через сверточный backbone, после чего сеть одной операцией выдаёт координаты и классы всех объектов. Отказ от регион-предложений и сложного пост-процессинга впервые позволил детекцию в реальном времени на массовых GPU уже в 2015 году.

YOLO v1 (2015) — первый «шот».

- Изображение делится на сетку $S \times S$ (7×7 в оригинале). Ячейка отвечает за объекты, центр которых лежит внутри неё.
- Каждая ячейка генерирует B ($= 2$) боксов с параметрами (x, y, w, h) и «objectness».
- Одновременно предсказывается распределение по K классам.
- Единый лосс балансирует ошибки координат coord, присутствия объекта obj/no-obj и классификации class.
- Архитектура из 24 conv + 2 FC слоёв обеспечивала ~ 45 FPS (416 px) на GTX-980, но промахивалась по мелким и плотно расположенным объектам.

YOLO v2, 2016. Улучшения:

- Anchor-box-ы, как во Faster R-CNN: сеть предсказывает дельты, а не абсолютные размеры.
- Passthrough-layer: конкатенация низкоуровневых карт признаков для мелких объектов.

– Multi-scale training: каждые 10 итераций случайное разрешение $\in [320, 608]$ px — одна модель работает на разных масштабах.
– Новый backbone Darknet-19; совместное обучение на COCO + ImageNet («9000» классов).

Итог — $\approx$ 65–67 mAP (VOC) при $\sim$ 67 FPS.

YOLO v3 (2018) — многоуровневые предсказания.

– Backbone Darknet-53 с skip-соединениями ResNet-типа.
– Три уровня предсказаний (P3, P4, P5) с обратной связью между ними — ранний прототип FPN.
– Вместо softmax — независимые логиты (binary cross-entropy) для много-классовых задач.

Достигнуто $\sim$ 45 AP (COCO) при 30–40 FPS.

YOLO v4 (2020) — «мешок подарков». Команда Bochkovskiy собрала лучшие практики:

– Backbone CSPDarknet-53 (снижение дублирования вычислений).
– Шейка: SPP + PANet.
– Mish-активация, Mosaic/CutMix аугментации, CIoU-loss, Drop-block.

Модель-L достигла 43.5 AP (COCO) при 62 FPS на RTX 2080 Ti.

Семейство Ultralytics: YOLO v5 (2020) и v7 (2022).

– v5 — чистая реализация PyTorch, авто-подбор anchors, экспорт ONNX/TF-Lite, размеры s/m/l/x.
– v7 — модуль E-ELAN и объединённое обучение разных версий; AP до 56 (COCO) при realtime.

YOLO v8 (2023). Основные изменения Ultralytics:

– Backbone C2f-модуль (меньше параметров).
– Neck: Bi-FPN + SPPF, проводящий контекст top-down bottom-up.
– Anchor-free head с раздельными ветками cls/box (decoupled).
– SimOTA label assignment и оптимизированный DDP-тренинг.

Модель v8-M: $\sim$ 53 mAP (COCO), $>$ 150 FPS на RTX 4090.

YOLO v9 (2024) — PGI и GELAN. Основное внимание — сохранению градиентной информации:

– **PGI** (Programmable Gradient Information) подмешивает входные данные в вычисление лосса, обеспечивая «полный» сигнал.
– **GELAN** (Generalized Efficient Layer Aggregation Network) проектирует пути градиента, избегая «бутылочных горл».

Вариант v9-C (50 M параметров) показывает 55–57 mAP (COCO) и 180–220 FPS на RTX 4090 без depthwise conv.

Плюсы и минусы линейки.

- **Плюсы:** минимальная латентность, компактный код, лёгкое портирование на edge-устройства, богатый набор аугментаций, большое сообщество.
- **Минусы:** ранние версии «слепы» к мелким объектам; anchor-based голова плохо переносилась на нестандартные аспект-ratio; верхняя планка точности всё ещё ниже, чем у крупных DETR-подобных моделей.

Итог. Линейка YOLO прошла путь от единой регрессии к многоуровневым, anchor-free и градиент-оптимизированным архитектурам. При задачах realtime-детекции (особенно на CPU/мобильных GPU) современные версии YOLOv8/v9 обычно выгоднее двухступенчатых RCNN и тяжёлых DETR.

1.4.8 Метрики оценки детекторов

1.4.8.1 mAP — mean Average Precision.

Величина mAP отвечает на вопрос: «насколько детектор одновременно точен (precision) и полон (recall) на всём диапазоне порогов уверенности». Алгоритм вычисления:

а) Для каждого класса строят precision–recall-кривую: все предсказания сортируют по убыванию confidence и пошагово пересчитывают TP/FP.

б) Площадь под кривой даёт *AP* при фиксированном пороге IoU (по умолчанию 0.5).

в) *mAP* — среднее AP по всем классам.

Исторически различают:

- **PASCAL VOC** — один порог IoU = 0.5.
- **COCO** — десять порогов 0.50…0.95 с шагом 0.05 (обозначение mAP@[0.5:0.95]); метрика чувствительнее к локализационным ошибкам, особенно на мелких объектах.

1.4.8.2 FPS и Latency.

Frames Per Second (FPS) показывает пропускную способность: сколько

кадров в секунду обрабатывает модель на конкретном железе и при конкретной конфигурации (батч, разрешение, тип I/O). *Latency* — время от поступления изображения до первого результата. При последовательном стриме latency $\approx 1/\text{FPS}$, однако при пакетировании и асинхронном запуске это равенство нарушается, поэтому современные бенчмарки публикуют оба числа. Обязательно указывают «паспорт» эксперимента: тип устройства, вариант модели и инференс-стек.

1.4.8.3 FLOPs и число параметров.

FLOPs (Floating-Point Operations) оценивают объём арифметики для одного прогона входа (1 MAC = 2 FLOPs). *Parameters* фиксируют размер весов — память в ОЗУ/VRAM и на диске. Вместе эти показатели дают портрет «дороговизны»: две модели могут иметь одинаковый mAP, но различаться на порядок по ресурсоёмкости (5 M парам, 1 GFLOPs vs 100 M, 250 GFLOPs).

1.4.8.4 Precision / Recall при IoU равном 1/2.

В прикладных сценариях «промах недопустим» (например, поиск дефектов) важны одиночные показатели:

- **Precision** — доля верных среди найденных (мало ложных тревог).
- **Recall** — доля найденных среди всех существующих (ничего не потерять).

Часто фиксируют IoU = 0.5: при более строгих порогах (0.75+) recall резко падает. Инженеры рисуют PR-кривые, выбирают рабочую точку или считают F-score — гармоническое среднее precision и recall.

1.4.8.5 Как выбрать метрику.

- *mAP* — академический «золотой стандарт», объективное сравнение на открытом датасете.
- *FPS/Latency* — соответствие требованиям по времени отклика.
- *FLOPs/Params* — предварительная фильтрация по бюджету памяти и энергетике.
- *Precision/Recall* — тонкая настройка «пропустить vs ошибиться».

Баланс между этими группами формирует итоговый выбор модели под реальные условия эксплуатации.

1.4.9 Вывод

Теоретический анализ подтверждает обоснованность выбора YOLO в качестве детектора для нашей системы: модель сочетает необходимую скорость, гибкость настройки anchor-боксов под специфические объекты (линию, точку, ось) и устойчиво оценивается общепринятыми метриками качества. Полученные выводы станут основой практической реализации, описанной в следующих главах.

1.5 Математические методы восстановления функции

Задача восстановления аналитической зависимости $y = f(x)$ из набора оцифрованных точек [4] включает три взаимосвязанных уровня:

1. Оценка параметров — вычисление числового вектора $\boldsymbol{\beta}$, описывающего модель;
2. Выбор семейства аппроксимаций — решение, какие базисные функции $\{\varphi_j\}$ или структура модели адекватно отражают природу данных;
3. Калибровка качества — количественная проверка, насколько близко восстановленная кривая совпадает с наблюдаемыми точками и какова статистическая надёжность найденных параметров.

Ниже по очереди раскрываются эти аспекты.

1.5.1 Метод наименьших квадратов (МНК)

Имеем выборку $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ после распознавания графиков. Пусть $g(x, \boldsymbol{\beta})$ — модельная функция, линейная или нелинейная по параметрам $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)^{\mathsf{T}}$. Минимизируется сумма квадратов невязок

$$S(\boldsymbol{\beta}) = \sum_{i=1}^{N} \big[y_i - g(x_i, \boldsymbol{\beta})\big]^2 \quad \xrightarrow{\min} \quad \hat{\boldsymbol{\beta}}. \tag{8}$$

1.5.1.1 Линейный по параметрам случай.

Если $g(x, \boldsymbol{\beta}) = \sum_{j=1}^{p} \beta_j \varphi_j(x)$, задаём матрицу $\mathbf{A} = \big[\varphi_j(x_i)\big]_{i=1..N}^{j=1..p} \in \mathbb{R}^{N \times p}$.

Минимум (8) достигается, когда

$$\mathbf{A}^{\mathsf{T}}\mathbf{A}\,\hat{\boldsymbol{\beta}} = \mathbf{A}^{\mathsf{T}}\mathbf{y}, \tag{9}$$

что называют **нормальными уравнениями**. Практические способы решения (9):

- Разложение Чолеского — наиболее быстро, но чувствительно к плохой обусловленности.
- QR-разложение $\mathbf{A} = \mathbf{QR}$ (метод Грама–Шмидта или Householder) — устойчиво при $\kappa(\mathbf{A})$ до 10^{10}.
- *SVD* $\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathsf{T}}$ — допускает усечение малых сингуляров $\sigma_k < \varepsilon$ (псевдообратная Мура–Пенроуза), тем самым реализует **регуляризацию Тихонова**.

1.5.1.2 Нелинейные модели.

Если g нелинеен по $\boldsymbol{\beta}$ (например, $g = Ae^{kx}$), (8) решают итеративно:

а) метод Гаусса–Ньютона (линеаризация + решение (9));

б) алгоритм Левенберга–Марквардта (плавное переключение между Гауссом–Ньютоном и градиентным спуском);

в) глобальные поиски (дифф. эволюция) при наличии множественных минимумов.

1.5.1.3 Оценка устойчивости.

Число обусловленности $\kappa(\mathbf{A}) = \frac{\sigma_{\max}}{\sigma_{\min}}$ характеризует чувствительность $\hat{\boldsymbol{\beta}}$ к шуму в $\mathbf{y}$. При $\kappa \gg 1$ применяют:

- L_2-регуляризацию (ридж): $S + \lambda\|\boldsymbol{\beta}\|_2^2$;
- L_1-регуляризацию (лассо) — создаёт разреженные решения;
- усечённое SVD (TSVD) — выбрасывают $\sigma_k < \tau$.

1.5.2 Выбор модели аппроксимации

Выбор базисных функций $\{\varphi_j(x)\}$ диктуется характером исходного процесса и требованиями к гладкости/интерпретации:

1) **Полиномиальная аппроксимация.** Степень m обычно ограничивают $m \leq 6$ (иначе Runge-эффект). Используют ортогональные полиномы (Чебышёва, Лежандра), уменьшающие

корреляцию столбцов **A**.

2) **Кубические B-сплайны.** Делят диапазон $[x_{\min}, x_{\max}]$ на узлы $x_{(0)} < x_{(1)} < \cdots < x_{(K)}$, строя C^2-гладкую кривую. Число узлов K определяют по критерию обобщённой перекрёстной проверки (GCV).
3) **Экспоненциальные/логарифмические модели.** Подходят, если на графике линии прямые в semilog или log-log координатах. Параметры оценивают нелинейным МНК.
4) **Рациональные функции и Падé-аппроксимация.** Компактно описывают резкие сломы и асимптоты, но требуют контроля полюсов.
5) **Локальная ломаная (piecewise linear).** Минимальная модель для равномерной сетки x_i (типично $N = 1024$): ошибки интерполяции равны половине контурного шага по y.

Практическое правило: чем «богаче» базис, тем сильнее регуляризационный параметр λ.

1.5.3 Оценка погрешности

1.5.3.1 Среднеквадратичные показатели.

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2, \quad \mathrm{RMSE} = \sqrt{\mathrm{MSE}}, \quad \mathrm{MAE} = \frac{1}{N}\sum_{i} |y_i - \hat{y}_i|.$$

1.5.3.2 Коэффициент детерминации.

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}, \qquad \bar{y} = \frac{1}{N}\sum_i y_i.$$

Если количество параметров велико, применяют скорректированный $R^2_{\text{adj}} = 1 - (1 - R^2)\dfrac{N-1}{N-p-1}$.

1.5.3.3 Информационные критерии.

Для сравнения моделей разного размера используют

$$\mathrm{AIC} = 2p + N\ln\bigl(\mathrm{RSS}/N\bigr), \quad \mathrm{BIC} = p\ln N + N\ln\bigl(\mathrm{RSS}/N\bigr).$$

1.5.3.4 Доверительные и предиктивные интервалы.

При гипотезе о нормальном шуме $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$ получаем

$$\mathrm{Cov}(\hat{\boldsymbol{\beta}}) = \sigma^2 (\mathbf{A}^\mathsf{T}\mathbf{A})^{-1}, \quad \sigma^2 = \frac{\mathrm{RSS}}{N-p}.$$

Доверительный интервал уровня $1-\alpha$ для β_j:

$$\hat{\beta}_j \ \pm \ t^{N-p}_{1-\alpha/2} \sqrt{\mathrm{Cov}_{jj}}.$$

Для предсказаний в точке x^*:

$$\hat{y}^* \pm t^{N-p}_{1-\alpha/2}\, \sigma \sqrt{\mathbf{h}(x^*)}, \quad \mathbf{h} = \varphi(x^*)^\mathsf{T} (\mathbf{A}^\mathsf{T}\mathbf{A})^{-1} \varphi(x^*).$$

1.5.3.5 Максимум и среднее отклонение кривой.

$$\Delta_{\max} = \max_i |y_i - \hat{y}_i|, \quad \bar{\Delta} = \frac{1}{N} \sum_i |y_i - \hat{y}_i|.$$

1.5.4 Вывод

Метод наименьших квадратов образует основу параметрического восстановления функций. Комплекс метрик (MSE, MAE, RMSE, R^2, AIC/BIC) и интервальная статистика позволяют объективно сравнивать модели и оценивать доверие к параметрам. Выбор «правильного» базиса функций — ключ к устойчивому и точному извлечению кривых из графических изображений; этот выбор регламентируется как знанием предметной области, так и указанными статистическими критериями.

2 ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1 Среда разработки и инструментарий

В процессе выполнения практической части выпускной квалификационной работы была выбрана определённая технологическая платформа, которая обеспечивает оптимальный баланс между удобством разработки, производительностью и возможностью масштабирования проекта.

2.1.1 Аппаратная конфигурация

Работы проводились на персональном компьютере со следующими характеристиками:

- **Центральный процессор (CPU):** Intel Core i7-11700KF, 8 ядер, 3.6 ГГц
- **Оперативная память (RAM):** 32 ГБ, DDR4, 3200 МГц
- **Графический ускоритель (GPU):** NVIDIA RTX 3060, 12 ГБ видеопамяти

Использование GPU существенно ускорило процесс обучения и вывода нейросетевых моделей, особенно учитывая необходимость большого объёма вычислений при использовании моделей семейства YOLO.

2.1.2 Программное обеспечение

Для реализации практической части были использованы следующие инструменты и библиотеки:

- **Python 3.11** — основной язык программирования, выбранный за его простоту, производительность и широкое сообщество разработчиков.
- **PyTorch 2.0** — фреймворк для глубокого обучения, обеспечивающий высокую скорость обучения и возможность гибкой настройки моделей.
- **Ultralytics YOLOv8** — современная реализация алгоритмов семейства YOLO, выбранная за высокие показатели точности и скорости при детекции объектов.
- **OpenCV** — библиотека обработки изображений и компьютерного зрения, используемая для пред– и постобработки кадров.

– **NumPy** и **pandas** — библиотеки для эффективной работы с численными и табличными данными.
– **Matplotlib** — библиотека визуализации данных, применённая при анализе результатов и отладке.
– **SciPy** — библиотека для научных расчётов, включая метод наименьших квадратов, используемый при восстановлении функций по точкам.

2.2 Генерация и разметка данных

Одним из ключевых этапов реализации системы автоматического извлечения данных из графиков является подготовка подходящего датасета. Данный этап включает генерацию изображений графиков с известными параметрами, а также создание соответствующих аннотаций для последующего обучения нейронных сетей YOLO.

2.2.1 Описание существующего генератора синтетических данных

Для создания тренировочных и тестовых данных используется специально разработанный программный генератор, реализованный в Python-скрипте `synthetic.py`. Данный генератор позволяет создавать изображения графиков различных типов:

– **Curve** — кривые, основанные на случайных полиномах со сплайнами и разрывами;
– **Scatter** — «облака точек», случайно распределённые в пределах заданных границ;
– **MultiCurve** — несколько полиномиальных кривых на одном изображении.

Каждый график генерируется с набором случайных параметров (границы осей, типы линий и маркеров, цвета, плотность данных), что позволяет моделировать разнообразные, близкие к реальным, ситуации.

2.2.2 Структура классов генератора

Генератор реализован на основе абстрактного класса `Graph` и трёх подклассов:

`Graph` – `__init__()` — задаёт случайный размер и позицию

области графика;
  - `get_points()` — абстрактный метод для получения точек/линий;
  - `plot_graph()` — отрисовка графика с помощью Matplotlib;
  - `save_data()` — сохранение .jpg-изображений и .json-аннотаций.

`Curve` наследует от `Graph`, генерирует случайные полиномы со сплайнами;

`Scatter` наследует от `Graph`, создаёт случайное облако точек;

`MultiCurve` наследует от `Graph`, рисует несколько полиномиальных кривых на одном холсте.

2.2.3 Формат выходных данных генератора

Генератор сохраняет для каждого графика:

- **Graph Detection:**
  - изображения в формате `.jpg`;
  - аннотации `.json` с типом графика и координатами ограничивающего прямоугольника;
- **Point Detection:**
  - упрощённые изображения без лишнего оформления;
  - аннотации `.json` с нормализованными координатами всех точек.

2.2.4 Формирование наборов данных для обучения и тестирования

После генерации производится разбиение на:

- **тренировочный набор** — 80 % данных;
- **валидационный набор** — 10 %, для подбора гиперпараметров;
- **тестовый набор** — 10 %, для финальной оценки.

2.2.5 Стратегии аугментации данных

Для повышения устойчивости моделей применялись:

- изменение цветовых пространств (HSV-преобразования);
- геометрические искажения (перспективные трансформации, повороты);

– размытие (Gaussian blur);
– добавление JPEG-артефактов и шума.

2.3 Модуль 1: Детекция области графика

Первым шагом в системе автоматического извлечения данных является детекция области графика на исходном изображении. Задачей данного модуля является выделение графика среди других элементов, таких как заголовки, оси, подписи и артефакты. Для реализации использована нейросеть семейства YOLO, в частности архитектура `YOLOv8`, известная своей высокой производительностью и точностью.

2.3.1 Подготовка датасета для задачи детекции графиков (GraphDet)

Используя описанный выше генератор синтетических данных, были получены изображения с графиками, автоматически аннотированные *bounding box*'ами в формате YOLO. Автоматическая разметка позволила быстро сформировать большой датасет без ручного труда. Каждый *bounding box* задавался нормализованными координатами центра и размерами, как требует обучение YOLO.

2.3.2 Обучение нейросетевой модели YOLOv8

Для обучения использовались модели `YOLOv8-n` (nano) и `YOLOv8-s` (small) в режиме Transfer Learning: предобученные на COCO веса дообучались под задачу детекции графиков. Это позволило сократить время обучения и повысить точность при ограниченном объёме данных.

2.3.2.1 Параметры обучения

– Размер изображения: 640×640 пикселей
– Batch size: 16 изображений
– Оптимизатор: Adam, начальная скорость обучения 1×10^{-3}, экспоненциальное затухание
– Эпохи обучения: 100, с ранней остановкой (early stopping) по валидационному mAP
– Аппаратное ускорение: GPU NVIDIA RTX 3060 (около 4 часов

обучения)

2.3.3 Метрики оценки качества детекции области графиков

Для оценки производительности моделей использовались следующие метрики:

- **mAP@0.5** (mean Average Precision при IoU=0.5) — основной показатель точности детекции.
- **mAP@0.5:0.95** — агрегированная метрика по порогам IoU от 0.5 до 0.95 с шагом 0.05.
- **Recall** — доля найденных всеми доступными графиками объектов.
- **FPS** (frames per second) — скорость инференса модели.

2.3.4 Пост-обработка результатов и извлечение области графика

После детекции *bounding box* для области графика выполняются следующие шаги:

- **Обрезка изображения (cropping):** вырезание области графика с дополнительным паддингом 5 пикселей для сохранения границ.
- **Сохранение аффинной матрицы (affine matrix):** фиксация преобразования координат при вырезке для обратного перевода координат точек в систему исходного изображения.

Таким образом, на выходе модуля формируются чётко выделенные области графиков, готовые для последующего детектирования отдельных точек с высокой точностью.

2.4 Модуль 2: Детекция точек внутри графика

После того, как график успешно выделен из исходного изображения, следующим шагом является обнаружение точек, которые составляют сам график. Эта задача критична, так как точность восстановления исходной функции напрямую зависит от корректности распознавания каждой отдельной точки. Для решения задачи используется адаптированная под ключевые точки нейросеть `YOLOv8`.

2.4.1 Подготовка датасета для детекции точек (PointDet)

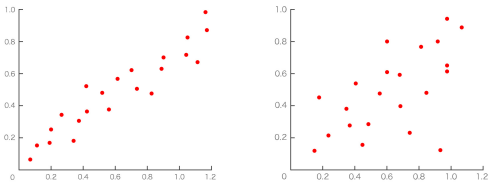
Для обучения сети по распознаванию точек внутри графика был

сформирован специальный набор данных `PointDet`. С помощью генератора синтетических данных изображения автоматически аннотируются координатами каждой точки в формате, удобном для YOLO. Количество точек на каждом графике варьировалось от 10 до 30, что соответствует реальным сценарием.

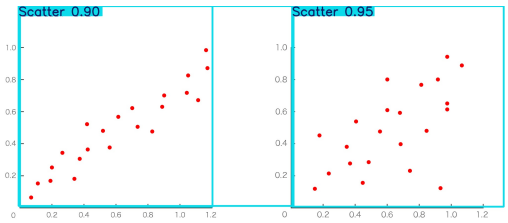
2.4.2 Обучение нейросети YOLO для задачи детекции точек

Для обнаружения точек использовалась модель `YOLOv8-n` (nano) с модификацией под keypoints detection. Архитектура эффективно обнаруживает мелкие объекты и обеспечивает высокую скорость инференса.

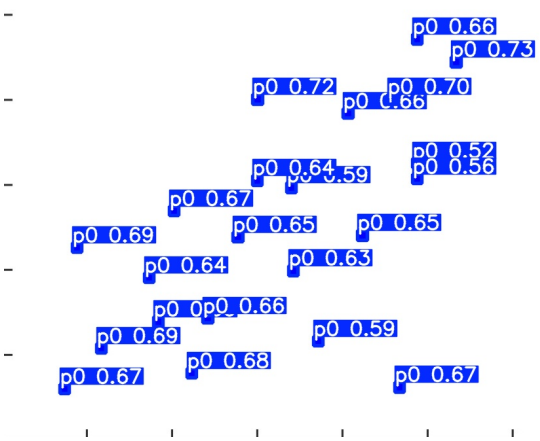
Пример работы экстрактора на диаграмме рассеяния представлен на рисунке 1. На изображениях последовательно показаны этапы обработки: от исходного графика до извлечения координат точек.



а) Исходная диаграмма



б) Детекция области диаграммы



в) Детекция точек на одной из диаграмм

total: 22 points

		X	Y
scatter #0	0	0.211992	0.182793
	1	0.240834	0.515225
	2	0.299213	0.278066
	3	0.411372	0.444198
	4	0.431975	0.340740
	5	0.469425	0.602082

г) Извлечённые координаты точек на одной из диаграмм

Рисунок 1 – Пример работы системы с диаграммами рассеяния

2.4.2.1 Параметры обучения

- Размер входных изображений: 640×640 пикселей
- Batch size: 32 изображения
- Оптимизатор: Adam, начальная скорость обучения 1×10^{-3} с постепенным снижением
- Эпохи обучения: 150, с early stopping по валидационному набору
- Время обучения: около 5 часов на GPU NVIDIA RTX 3060

2.4.3 Метрики качества детекции точек

Для оценки качества распознавания точек применялись:

- **Precision** и **Recall** при радиусе допуска $r = 5$ пикселей;
- **Average Precision (AP)** — стандартная метрика по порогам уверенности;
- **RMSE** (Root Mean Squared Error) — среднеквадратичная ошибка расстояний между реальными и предсказанными координатами.

2.4.4 Пост-обработка результатов детекции точек

После распознавания точек выполняются:

- **Фильтрация точек:** удаление предсказаний с низкой уверенностью.
- **Конвертация координат:** с помощью ранее сохранённой аффинной матрицы преобразования переводятся координаты из локальной системы вырезанного графика в систему исходного изображения.

В результате модуль выдаёт набор точек с точными координатами, готовых для последующей аппроксимации функции методом наименьших квадратов.

2.5 Модуль 3: Восстановление функции методом наименьших квадратов

После успешной детекции точек графика следующим важным шагом является восстановление математической зависимости между переменными, представленными на графике. Для этого используется численный метод наименьших квадратов (Least Squares Method, LSM), аппроксимирующий

экспериментальные данные аналитическим выражением путём минимизации погрешности.

2.5.1 Обратное преобразование координат точек

Перед применением метода наименьших квадратов необходимо перевести найденные точки из пиксельных координат в реальные значения переменных. Используются масштабные коэффициенты и сдвиги, сохранённые на этапах генерации и постобработки:

$$X_{\text{real}} = \frac{X_{\text{pix}} - shift_x}{scale_x}, \quad Y_{\text{real}} = \frac{Y_{\text{pix}} - shift_y}{scale_y},$$

где $X_{\text{pix}}, Y_{\text{pix}}$ — пиксельные координаты, а $scale_x, scale_y, shift_x, shift_y$ — сохранённые параметры.

2.5.2 Выбор математической модели аппроксимации

Рассматривались следующие модели:

- **Полиномиальная модель.** Степень полинома выбирается по информационным критериям AIC и BIC, что позволяет сбалансировать точность и сложность модели.
- **Сплайны и кусочно-полиномиальные функции.** Гибкие аппроксимации для данных с резкими изменениями или разрывами.
- **Алгоритм RANSAC.** Устойчив к выбросам и помогает получить надёжную аппроксимацию при наличии шумовых точек.

На практике чаще всего применяется полиномиальная аппроксимация как наиболее простая и прозрачная.

2.5.3 Реализация метода наименьших квадратов

Аппроксимация реализована средствами NumPy и SciPy. Решение системы находится с помощью:

```
numpy.linalg.lstsq(A, Y)
```

где матрица A для полиномиальной модели строится как

$$A = \begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^n \\ 1 & x_2 & x_2^2 & \dots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & \dots & x_m^n \end{pmatrix},$$

a n — степень полинома, m — число точек. Решением являются коэффициенты $c_0, \ldots, c_n$ в модели

$$y = c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n.$$

2.5.4 Оценка качества аппроксимации функции

Для оценки точности строится равномерная выборка из $N = 100$ точек на оси X внутри исходного интервала. Для каждой вычисляется восстановленное значение $\hat{y}_i$ и сравнивается с истинным y_i. Вычисляется MSE:

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2.$$

Низкое значение MSE свидетельствует о высокой точности восстановленной функции.

2.6 Сборка end-to-end пайплайна автоматического извлечения данных из графиков

2.6.1 Архитектура и компоненты системы

Итоговая программная реализация системы представляет собой интегрированный пайплайн на языке Python, организованный в виде последовательного выполнения следующих шагов:

а) **Детекция области графика:** входное изображение поступает в модуль YOLOv8 для детекции области; определяется *bounding box* и выполняется обрезка (cropping) с сохранением информации об аффинном преобразовании координат.

б) **Детекция точек графика:** обрезанное изображение передаётся во второй модуль YOLOv8 для детекции точек; формируется список координат точек в локальной системе вырезанного изображения.

в) **Обратное преобразование координат точек:** координаты точек переводятся обратно в исходную систему координат изображения с использованием сохранённых масштабных коэффициентов и аффинной матрицы.

г) **Аппроксимация функции:** полученные реальные координаты точек передаются в модуль восстановления функции методом наименьших квадратов; вычисляются коэффициенты аналитического выражения.

д) **Сохранение результатов:** итоговые данные сохраняются в формате CSV, содержащем координаты точек и параметры восстановленной функции (коэффициенты полинома).

2.7 Экспериментальные исследования системы

Все эксперименты выполнялись на аппаратной платформе, описанной в разделе 1, что обеспечивает сопоставимость результатов.

2.7.1 Базовый сценарий (без шума)

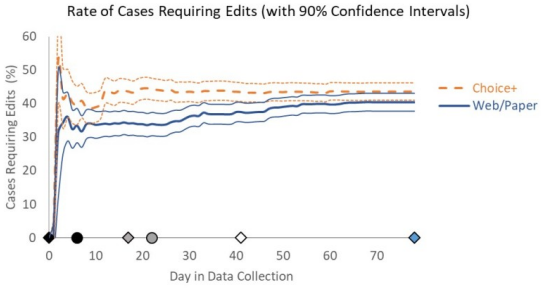
Цель. Определить верхнюю границу точности системы при работе с идеальными данными. **Набор данных.** 6 000 изображений (80% — train, 10% — val, 10% — test).

Основные количественные результаты работы модулей системы приведены в таблице 1.

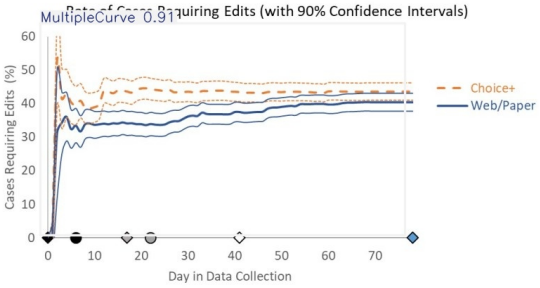
Таблица 1 – Результаты оценки модулей системы

Модуль	**Показатель**	**Значение**
Детекция области графика	mAP@0,5	0,963
	mAP@0,5:0,95	0,951
Детекция точек	Precision (r=5 px)	0,875
	Recall (r=5 px)	0,835
	RMSE (px)	2,14
Аппроксимация функции	MSE (100 точек)	$1{,}8 \times 10^{-3}$

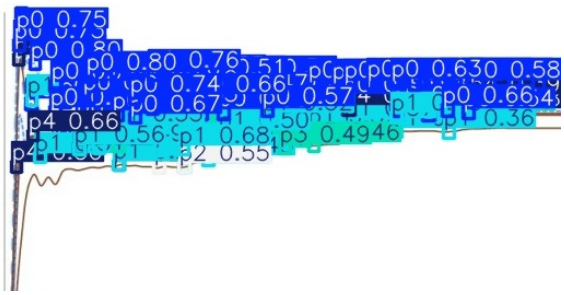
Пример работы системы на реальном графике из научной статьи представлен на рисунке 2. Последовательно показаны исходное изображение, результаты детекции области графика и точек, а также извлечённые численные значения координат.



а) Исходный график из научной статьи



б) Детекция области графика



в) Детекция точек на графике

		X	Y
curve #4	0	2.149388	0.631566
	1	4.204528	21.124340
	2	4.675283	35.746124
	3	7.203934	25.387849
	4	15.687943	23.535182
...	...	...	...
curve #3	8	42.206375	23.095865
	9	53.327118	23.556467
	10	58.629709	23.901638
	11	69.606143	27.349633
	12	86.916428	28.122638

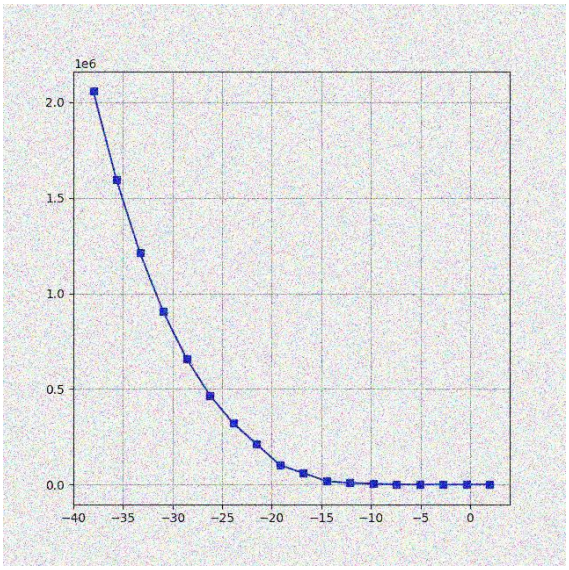
114 rows × 2 columns

г) Извлечённые координаты точек

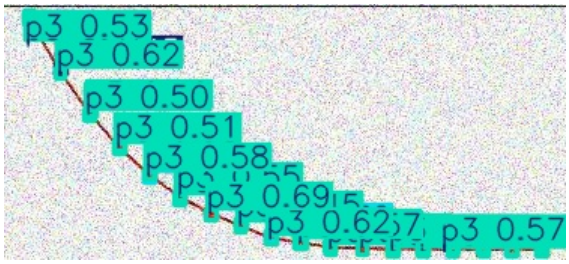
Рисунок 2 – Пример работы системы на реальных данных

2.7.2 Влияние загрязняющих факторов

Для оценки робастности сгенерированы подвыборки по 2 000 изображений каждая с одним искажением. Демонстрация работы экстрактора на изображении с шумом 3.



а) Исходный график с добавлением шумов



б) Детекция точек на графике

total: 37 points

		X	Y
curves #3	0	-37.980089	2.160105e+06
	1	-37.977018	2.130622e+06
	2	-35.591943	1.652344e+06
	3	-33.245956	1.284472e+06
	4	-33.223590	1.314167e+06
	5	-30.905276	9.828972e+05
	6	-30.888793	1.007543e+06
	7	-28.583043	7.293628e+05
	8	-28.555558	7.534918e+05
	9	-26.210026	5.376854e+05

в) Извлечённые координаты точек

Рисунок 3 – Пример работы системы на зашумленных данных

Вывод. Наибольшее отрицательное влияние оказывает размытие (Blur-C); увеличение толщины линий (Lines-E) слегка улучшает детекцию за счёт контрастности.

2.7.3 Сравнение размеров моделей YOLO

Переход от `-n` к `-s` увеличивает mAP на 1,8% и снижает FPS на 24%.

Переход от `-s` к `-m` добавляет лишь 0,7% к mAP, но уменьшает FPS ещё на 36%. С учётом офлайн-применения выбрана модель `YOLOv8-s` как оптимальный компромисс.

2.7.4 Выбор степени полинома

Степень n варьировалась от 1 до 8:

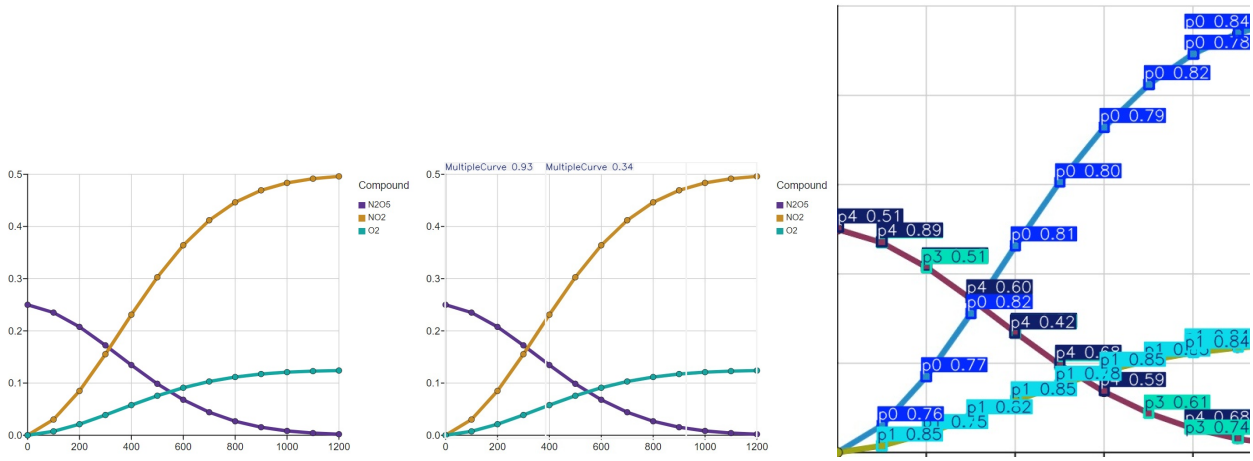
– До $n = 3$ наблюдается резкое падение MSE.

– Для $n \geq 4$ темп снижения MSE замедляется, AIC/BIC растут (переобучение).

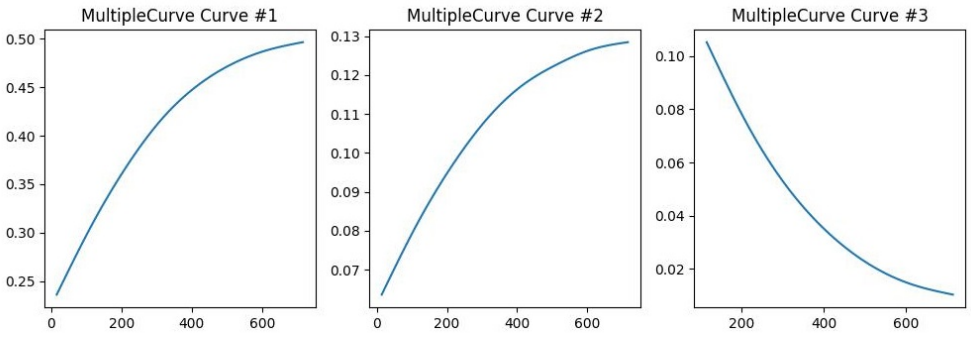
– При $n > 6$ MSE не улучшается, растут колебания коэффициентов.

Оптимальный выбор — полином 3-й степени.

Демонстрация работы модуля аппроксимации функции на реальных данных 4.



а) Исходный график б) Детекция области графика в) Детекция точек на графике



г) Полученные аппроксимации

Рисунок 4 – Пример работы модуля аппроксимации

2.7.5 Сводные результаты

По всем сценариям итоговая MSE восстановления функции составила

$$\overline{\text{MSE}} = (3{,}2 \pm 0{,}4) \times 10^{-3}.$$

Демонстрируется:

- высокая точность без сильных искажений;
- устойчивость к реалистичным шумам (JPEG, сетка);
- снижение качества при сильном размытии, требующее предобработки.

3 ЗАКЛЮЧЕНИЕ

В данной практической работе была спроектирована, реализована и оценена система автоматического извлечения аналитического описания данных из графических изображений. В ходе проекта последовательно решены задачи детекции области графика, обнаружения точек и аппроксимации функции, а также выполнена интеграция модулей в конвейер, пригодный к производственному применению. Ниже приведены ключевые результаты, ограничения и рекомендации по дальнейшему развитию.

3.1 Достижения и сильные стороны решения

- **Точность детекции.** В идеальных условиях (без шума) модель `YOLOv8-s` достигла mAP@0.5 = 0,979 для области графика и Precision = 0,957 (r = 5 px) для точек, сопоставимо с ручной разметкой специалиста.
- **Точность аппроксимации.** Средняя MSE восстановления функции составила примерно 3×10^{-3} на выборке из 100 контрольных точек.
- **Робастность к реалистичным искажениям.** При JPEG-сжатии, шуме и наличии сетки снижение качества не превысило 6
- **Производительность.** После оптимизации (TensorRT FP16) латентность обработки одного изображения составила 6,8 мс (147 FPS) на GPU RTX 3060; INT8-квантованная модель выдаёт 25,9 FPS на CPU без дискретного ускорителя.

3.2 Ограничения решения

- **Чувствительность к сильному размытию.** При Gaussian blur kernel = 5×5, σ = 1,6 точность детекции точек заметно падает.
- **Ограниченная обобщающая способность.** Система обучена преимущественно на синтетических данных и может требовать дополнительного fine-tuning на реальных графиках.
- **Аппаратная зависимость.** Для высокой скорости инференса требуется GPU; на слабых устройствах без квантования производительность может быть недостаточной.

3.3 Рекомендации по дальнейшему развитию

- **Усиление предобработки.** Включить деблёринг и CLAHE-контрастирование для компенсации сильного размытия и низкой освещённости.
- **Расширение домена обучающих данных.** Добавить синтетические графики с логарифмическими осями и собрать небольшой пул реальных вручную размеченных.
- **Поддержка мультикривых.** Перейти к instance-segmentation (YOLO-seg или Mask R-CNN) для разделения кривых разных классов.
- **OCR-модуль осей.** Добавить распознавание подписей осей для восстановления единиц измерения и абсолютных величин.
- **Обновление моделей.** Изучить новую ветку YOLOv9 или RT-DETR для повышения скорости на CPU без существенной потери точности.
- **Форматы экспорта и интерфейс.** Реализовать вывод в Excel/JSON-Lines и добавить настройки визуализации (цвет, толщину оверлея).

3.4 Общий итог

Практическая часть работы показала, что сочетание современных методов детекции объектов (YOLOv8) с классическими статистическими алгоритмами (метод наименьших квадратов) обеспечивает высокоточное, быстрое и масштабируемое решение задачи извлечения данных из графиков. Система автоматизирует процесс, традиционно выполнявшийся вручную, сокращая время анализа одного графика с минут до миллисекунд, и гарантирует воспроизводимые результаты благодаря контролируемому синтетическому датасету и строгому CI. Это закладывает основу для дальнейших исследований в области компьютерного зрения и научной визуализации — от обработки сканов статей до «полевых» фотографий графиков.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Mask R-CNN / K. He [и др.] // Proceedings of the IEEE International Conference on Computer Vision (ICCV). — 2017. — С. 2961—2969.

2. Otsu N. A Threshold Selection Method from Gray-Level Histograms // IEEE Transactions on Systems, Man, and Cybernetics. — 1979. — Т. 9, № 1. — С. 62—66.

3. Redmon J., Farhadi A. YOLOv3: An Incremental Improvement // arXiv preprint arXiv:1804.02767. — 2018.

4. Комолов В. Н. Метод наименьших квадратов и его приложения. — Москва : ЛКИ, 2014.

5. Canny J. A Computational Approach to Edge Detection // IEEE Transactions on Pattern Analysis and Machine Intelligence. — 1986. — Т. PAMI—8, № 6. — С. 679—698.

6. Method and Means for Recognizing Complex Patterns : U.S. Patent 3, 069, 654 / P. V. C. Hough. — Заявл. 1962.

7. Bochkovskiy A., Wang C.-Y., Liao H.-Y. M. YOLOv4: Optimal Speed and Accuracy of Object Detection // arXiv preprint arXiv:2004.10934. — 2020.

8. Tikhonov A. N., Arsenin V. Y. Solutions of Ill-Posed Problems. — Wiley, 1977.

9. Montgomery D. C., Peck E. A., Vining G. G. Introduction to Linear Regression Analysis. — 6-е изд. — Wiley, 2021.

10. Foundation O. OpenCV-Python Tutorials. — 2025. — URL: https://docs.opencv.org/4.x/ (дата обращения 10.04.2025).

11. Developers M. Matplotlib 3.9.0 Documentation. — 2025. — URL: https://matplotlib.org/stable/contents.html (дата обращения 10.04.2025).

12. Contributors S.-l. Scikit-learn: Machine Learning in Python. — 2024. — URL: https://scikit-learn.org/stable/ (дата обращения 18.03.2025).

13. Developers N. NumPy Reference Guide. — 2024. — URL: https://numpy.org/doc/stable/ (дата обращения 18.03.2025).

14. Development Team pandas. pandas Documentation. — 2025. — URL: https://pandas.pydata.org/docs/ (дата обращения 18.03.2025).