

Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
Кафедра «Информационные системы»

УТВЕРЖДАЮ
Заведующий кафедрой ИС
_____ А.В. Раскина
подпись
« _____ » _____ 2025 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Распознавание объектов на дороге в реальном времени для управления
беспилотным автомобилем

09.04.01 Информатика и вычислительная техника

09.04.01.13 Инженерия искусственного интеллекта

Руководитель	_____	<u>доцент, канд.техн.наук</u>	А.В. Пятаева
	подпись, дата	должность, учёная степень	
Выпускник	_____		Д.М. Паранина
	подпись, дата		
Рецензент	_____	<u>доцент, канд.техн.наук</u>	Н.А. Сергеева
	подпись, дата	должность, учёная степень	
Нормоконтролер	_____		Ю.В. Шмагрис
	подпись, дата		

Красноярск 2025

Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
Кафедра «Информационные системы»

УТВЕРЖДАЮ
Заведующий кафедрой ИС
_____ А.В. Раскина
подпись
«_____» _____ 2025 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
в форме магистерской диссертации

Красноярск 2025

Студенту Параниной Дарье Михайловне

Группа: КИ23-01-13М Направление: 09.04.01 «Информатика и вычислительная техника»

Тема выпускной квалификационной работы: «Распознавание объектов на дороге в реальном времени для управления беспилотным автомобилем».

Утверждена приказом по университету №7302/С от 18.04.2025г.

Руководитель ВКР: А.В. Пятаева, к.т.н., доцент кафедры «Системы искусственного интеллекта» ИКИТ СФУ.

Исходные данные для ВКР: Требования к точности и производительности моделей, рекомендации руководителя, учебные материалы.

Перечень разделов ВКР: Введение, теоретические аспекты нейронных сетей в задаче распознавания объектов, проектирование нейросетевого решения для детекции дорожных объектов в реальном времени, дообучение моделей распознавания объектов в реальном времени на дорожных сценах и сравнительный анализ их эффективности, заключение.

Перечень графического материала: презентация, выполненная в Microsoft PowerPoint.

Руководитель ВКР

подпись, дата

А.В. Пятаева

Задание принял к исполнению

подпись, дата

Д.М. Паранина

«____» _____ 2025г

РЕФЕРАТ

Выпускная квалификационная работа по теме «Распознавание объектов на дороге в реальном времени для управления беспилотным автомобилем» содержит 93 страницы текстового документа, 26 иллюстраций, 6 таблиц, 5 приложений, 116 использованных источников.

ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ, КОМПЬЮТЕРНОЕ ЗРЕНИЕ, YOLO, RT-DETR, ТРАНСФОРМЕРЫ, БЕСПИЛОТНОЕ УПРАВЛЕНИЕ АВТОМОБИЛЕМ, РАСПОЗНАВАНИЕ ОБЪЕКТОВ В РЕАЛЬНОМ ВРЕМЕНИ.

Целью настоящей магистерской диссертации является исследование и практическое применение моделей глубокого обучения для разработки решений, способных распознавать дорожные объекты в видеопотоке в реальном времени.

Задачи исследования включают в себя:

1. Изучить теоретические основы нейронных сетей и исследования применению глубокого обучения к распознаванию объектов на дороге;
2. Проанализировать современные архитектуры глубоких нейронных сетей и инструменты для их обучения и оценки в задачах распознавания объектов в реальном времени.
3. Разработать методологию дообучения выбранных моделей на специализированном датасете дорожных сцен и выполнить их адаптацию;
4. Провести тестирование и сравнительный анализ точности и производительности дообученных моделей.

В рамках работы был разработан и применен алгоритм дообучения моделей глубокого обучения на специализированных данных дорожных сцен для адаптации к задаче распознавания дорожных объектов. Разработанное решение имеет потенциал использования в качестве компонента системы управления беспилотным автомобилем, что в перспективе позволит сделать использование транспорта более безопасным, удобным, экологичным и доступным.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	4
1 Нейронные сети в задаче распознавания объектов на дороге: теоретические аспекты и обзор существующих решений	7
1.1 Нейронные сети: базовые понятия, CNN и трансформеры	7
1.1.1 Базовые принципы и типы нейронных сетей	7
1.1.2 Принцип работы сверточных нейронных сетей	13
1.1.3 Принцип работы трансформеров	17
1.2 Методы обучения нейронных сетей и обработки данных	23
1.3 Методы оценки моделей распознавания объектов в реальном времени	27
1.4 Литературный обзор работ по распознаванию объектов на дороге с применением нейросетей	33
Выводы по первой главе	39
2. Модели, данные и инструменты для детекции дорожных объектов в реальном времени	41
2.1 Обзор предобученных моделей детекции объектов в реальном времени	41
2.2.1 Архитектура и особенности моделей YOLO	41
2.2.2 Архитектура и особенности моделей RT-DETR	44
2.2 Описание и предобработка данных дорожных сцен	46
2.3 Программные средства и библиотеки для реализации решения	55
Выводы по второй главе	57
3 Дообучение моделей и оценка их эффективности в задаче дорожной детекции в реальном времени	58
3.1 Дообучение моделей YOLOv8n и RT-DETR на датасете BDD100K ..	58
3.2 Сравнительный анализ эффективности дообученных моделей	65
3.3 Демонстрация моделей и рекомендации для дальнейших исследований	73
Выводы по третьей главе	77
ЗАКЛЮЧЕНИЕ	79
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	82
ПРИЛОЖЕНИЕ А Отчет о проверке на заимствование	96
ПРИЛОЖЕНИЕ Б Преобразование классов и очистка данных BDD100K	97
ПРИЛОЖЕНИЕ В Oversampling и undersampling тренировочных данных	100
ПРИЛОЖЕНИЕ Г Преобразование тренировочных данных BDD100K в форматы YOLO и COCO	103
ПРИЛОЖЕНИЕ Д Создание кастомного датасета для дообучения RT-DETR ..	106

ВВЕДЕНИЕ

С развитием технологий и усилением внимания к вопросам безопасности дорожного движения, беспилотные автомобили становятся важным этапом в эволюции транспорта. Одним из ключевых компонентов беспилотных транспортных средств является алгоритм распознавания объектов в реальном времени на основе данных с камеры автомобиля.

Сегодня для создания подобных алгоритмов, как и для решения большинства задач компьютерного зрения, широко применяются искусственные нейронные сети – математические модели, основанные на принципах работы мозга, способные эффективно анализировать визуальную информацию.

В рамках работы был разработан и применен алгоритм дообучения моделей глубокого обучения на специализированных данных дорожных сцен для адаптации моделей к задаче распознавания дорожных объектов в реальном. Разработанное решение может быть использовано в качестве компонента системы управления беспилотным автомобилем, что в перспективе позволит сделать использование транспорта более безопасным, удобным, экологичным и доступным.

Актуальность настоящей работы обусловлена необходимостью адаптации современных моделей глубокого обучения к условиям дорожной инфраструктуры и получения нейросетевого решения, пригодного для надежной интеграции в практические системы управления беспилотными транспортными средствами.

Объектом исследования являются нейросетевые методы распознавания объектов в дорожной среде в реальном времени.

Предметом исследования является дообучение и адаптация моделей глубокого обучения для распознавания дорожных объектов в условиях реальной дорожной инфраструктуры.

Объектом разработки являются алгоритм дообучения и дообученные нейросетевые модели (YOLOv8 и RT-DETR), предназначенные для применения в системах распознавания объектов на дороге в реальном времени и возможной интеграции в системы автономного вождения или помощи водителю.

Основная идея работы заключается в применении моделей глубокого обучения, дообученных на специализированном датасете дорожных сцен для создания программного обеспечения, способного эффективно распознавать объекты на дороге в реальном времени. В работе рассматриваются как проверенные временем архитектуры сверточных нейронных сетей (YOLO), так и относительно новые подходы, основанные на трансформерах, которые в задачах беспилотного транспорта применяются относительно недавно и пока мало исследованы в российской практике. Это позволяет провести сравнительный анализ и определить, могут ли трансформерные модели стать перспективной альтернативой сверточным в задачах компьютерного зрения для беспилотного транспорта.

Целью настоящей магистерской диссертации является исследование и практическое применение моделей глубокого обучения для разработки решений, способных распознавать дорожные объекты в видеопотоке в реальном времени.

Задачи исследования включают в себя:

1. Изучить теоретические основы нейронных сетей и проанализировать существующие исследования применения глубокого обучения для распознавания объектов на дороге;
2. Проанализировать современные архитектуры глубоких нейронных сетей и инструменты для их обучения и оценки в задачах распознавания объектов в реальном времени.
3. Разработать методологию дообучения выбранных моделей на специализированном датасете дорожных сцен и выполнить их адаптацию;
4. Провести тестирование и сравнительный анализ точности и производительности дообученных моделей.

Для достижения цели использовались метод дообучения нейронных сетей (fine-tuning), методы предобработки данных (андерсемплинг, оверсемплинг, нормализация, масштабирование), методы стабилизации обучения и оценки точности и производительности моделей (mAP@50, mAP@[0.5:0.95], latency, FPS, вес модели). В качестве инструментальных средств использованы: язык программирования Python, фреймворки для загрузки и дообучения нейронных сетей – Ultralytics YOLO и HuggingFace Transformers, а также другие вспомогательные библиотеки. Среда разработки – Jupyter Notebook.

Научная новизна работы:

1) Впервые проведено дообучение трансформерной модели RT-DETR на датасете дорожных сцен BDD100K, а также зафиксированы результаты дообучения модели YOLOv8n на данном датасете, которые ранее не были представлены в научной литературе.

2) Выполнен сравнительный анализ эффективности сверточной модели YOLOv8n и трансформерной модели RT-DETR-R101-VD в задаче распознавания дорожных объектов в реальном времени, что ранее не проводилось в контексте применения к датасету BDD100K.

Значение для теории заключается в расширении области применения трансформеров в задачах компьютерного зрения в реальном времени и уточнении их позиций в качестве альтернативы традиционным сверточным архитектурам.

Значение для практики заключается в том, что результаты работы могут быть использованы для повышения безопасности и эффективности беспилотного вождения, а также имеют потенциал интеграции в системы помощи водителю и интеллектуальные видеорегистраторы.

Все основные результаты работы получены автором лично. Использовался открытый датасет BDD100K и модели YOLOv8n и RT-DETR-R101-VD, распространяемые под открытыми лицензиями.

1 Нейронные сети в задаче распознавания объектов на дороге: теоретические аспекты и обзор существующих решений

1.1 Нейронные сети: базовые понятия, CNN и трансформеры

1.1.1 Базовые принципы и типы нейронных сетей

Нейронная сеть – это модель машинного обучения, основанная на принципах, вдохновлённых структурой и функционированием человеческого мозга. Она состоит из взаимосвязанных узлов (нейронов), которые могут обучаться на данных и принимать решения. Эти системы используются для решения множества задач, таких как распознавание образов, классификация, прогнозирование и других задач, связанных с обработкой больших объёмов данных.

Авторы статьи о принципах работы и архитектуре нейронных сетей предлагают следующее определение: «Нейронные сети – это искусственный интеллект, который представляет собой вид машинного обучения, посредством которого программное обеспечение компьютера способно имитировать человеческий мозг. По принципу передачи между собой сигналов нейронов человеческого мозга, вычислительные элементы нейронной сети способны обмениваются информацией» [1].

Нейронные сети начали активно развиваться ещё в середине XX века, однако их широкое применение стало возможным лишь с развитием вычислительных мощностей и доступом к большим данным в последние десятилетия [2]. Сегодня они широко используются в различных сферах для решения множества задач, включая распознавание образов [3].

Нейронные сети состоят из простых элементов – искусственных нейронов, которые можно рассматривать как математические модели, имитирующие работу биологических нейронов. Каждый нейрон в сети получает входные

сигналы через «синапсы», а выходные сигналы передает через «аксон» (см. Рисунок 1) [3]. В биологических нейронных сетях уровень активности нейрона определяется балансом между возбуждающими и тормозными синаптическими воздействиями. Превышение порогового уровня возбуждения инициирует потенциал действия; преобладание тормозных влияний подавляет активность нейрона [4].

В искусственных нейронных сетях входы и выходы представлены числовыми значениями, а активация нейрона зависит от величины суммарного входа и функции активации. При высокой активации выходное значение существенно отличается от нуля, тогда как при подавлении оно стремится к нулю или принимает строго нулевое значение, в зависимости от выбранной функции активации.

Рассмотрим устройство искусственного нейрона более подробно [3]. Как изображено на рисунке 1, каждый вход нейрона x_i имеет соответствующий весовой коэффициент ω_i , аналогичный силе синаптической связи в биологических нейронных сетях.

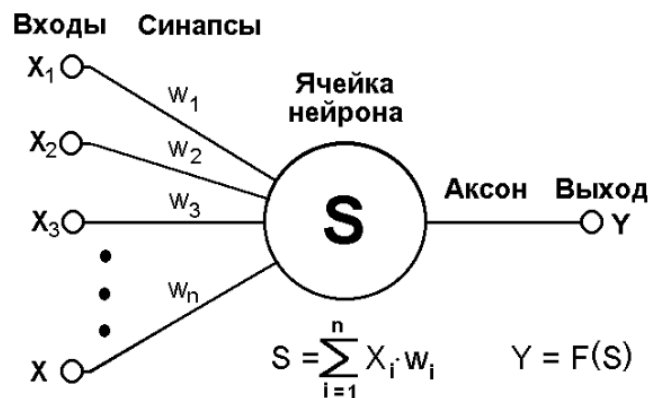


Рисунок 1 – Схематичное представление нейрона [3]

На первом этапе вычисляется взвешенная сумма входов по формуле (1):

$$S = \sum_{i=1}^n x_i \cdot \omega_i \quad (1)$$

Затем к полученному значению применяется функция активации $f(s)$, определяющая выход нейрона – формула (2):

$$y = f(s) \quad (2)$$

Таким образом, выход y зависит от совокупного входного сигнала и выбранной функции активации, которая задаёт характер преобразования линейной комбинации входов в выходной отклик. Выбор функции активации существенно влияет на способность нейрона моделировать нелинейные зависимости.

Связи между нейронами играют ключевую роль в передаче и обработке информации. Выход одного нейрона служит входом для нейронов следующего слоя, что формирует каскадную архитектуру обработки сигналов. Совокупность нейронов, выполняющих однотипные операции, образует слой сети. Каждый нейрон слоя принимает данные от предыдущего слоя, применяет функцию активации и передаёт результат далее.

В дополнение к входным данным нейроны, как правило, получают параметр смещения ($bias$), не зависящий от входа [5]. Его назначение – сместить функцию активации, позволяя модели аппроксимировать зависимости, не проходящие через начало координат. Как видно из формулы (3), смещение b добавляется к сумме произведений входных данных и весов. Его значение оптимизируется вместе с весами во время обучения:

$$y = f(\sum_{i=1}^n x_i \cdot \omega_i + b) \quad (3)$$

Принято классифицировать слои на входной, скрытые и выходной [2]. Входной слой – первый, в нем не происходит обучения, он лишь принимает данные и передает их дальше. Выходной слой – это последний слой, выдающий

финальный результат обучения, то есть его выход является ответом нейронной сети на поставленную задачу.

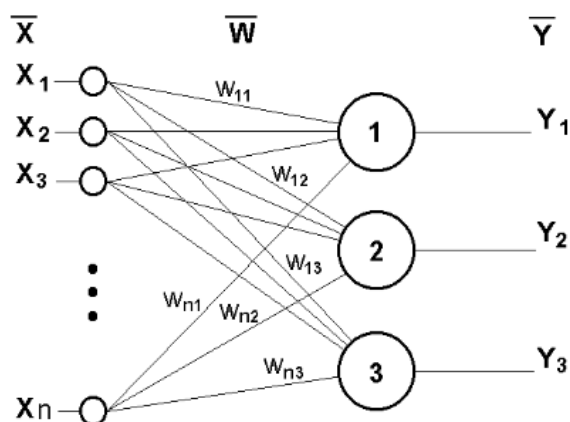


Рисунок 2 – Схема однослойного персептрона [3]

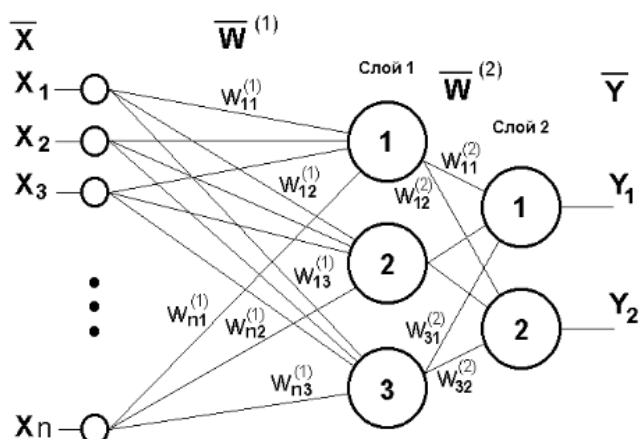


Рисунок 3 – Схема двухслойного персептрона [3]

Скрытые слои находятся между входным и выходным. Они обрабатывают данные, выявляют скрытые закономерности. Технически, в нейронной сети может быть только входной и выходной слой, а скрытые слои отсутствовать. В таком случае все вычисления – весов, сумм и активаций, выполняются на уровне выходного слоя. Такой архитектурой, например, является классический (однослойный) персептрон (Рисунок 2), который был придуман в 1957 году Фрэнком Розенблаттом [6]. Модель с такой архитектурой может решать только простые линейно разделимые задачи.

Если в нейронной сети больше одного слоя, её принято считать многослойной [2]. Пример многослойного персептрона представлен на рисунке 3.

Функцией активации персептрона является пороговая функция (шаговая) (формула 4). Можно сказать, что это самая примитивная активационная функция. Принцип её функционирования заключается в следующем: «Если взвешенная сумма всех входных сигналов окажется выше определенного порогового значения, то нейрон окажется возбужденным, то есть будет передавать сигнал всем нейронам, с которым соединяется в следующем слое» [7]:

$$f_{\text{threshold}}(x) = \begin{cases} 1, & x \geq \theta \\ 0, & x < \theta \end{cases} \quad (4)$$

В современных нейронных сетях чаще всего используют более сложные активационные функции [8].

Sigmoid – выдаёт значение от 0 до 1. Используется, например, в задачах бинарной классификации (формула 5):

$$f_{\text{sigmoid}}(x) = \frac{1}{1 + e^{-x}} \quad (5)$$

ReLU (Rectified Linear Unit) – возвращает 0, если вход меньше или равен 0, и значение x , если $x > 0$. Является одной из самых популярных функций в скрытых слоях современных сетей (формула 6):

$$f_{\text{relu}}(x) = \max(0, x) \quad (6)$$

Tanh (гиперболический тангенс) – выдаёт значение от -1 до 1. Центрирована вокруг нуля, что может быть полезно при обучении (формула 7):

$$f_{\tanh}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\sigma(2x) - 1 \quad (7)$$

Softmax – преобразует вектор чисел в вероятностное распределение (все значения от 0 до 1 и в сумме дают 1). Применяется на выходном слое при многоклассовой классификации (формула 8):

$$f_{\text{softmax}}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}} \quad \text{для } x = (x_1, x_2, \dots, x_n) \quad (8)$$

Другими отличительными чертами персептрона являются его полносвязность и прямая направленность. Рассмотрим эти понятия поподробнее.

Полносвязная нейронная сеть (или fully connected neural network, сокращённо FCN) – это тип нейронной сети, в которой каждый нейрон в одном слое связан с каждым нейроном в следующем слое [9]. Это значит, что для каждой пары слоёв существует полное соединение между всеми нейронами. Примером полносвязных нейронных сетей могут послужить персептроны на Рисунках 2 и 3. Многие более сложные архитектуры (например, сверточные или рекуррентные нейронные сети) включают полносвязные слои на выходе для принятия решения.

Прямонаправленная нейронная сеть (англ. Feedforward Neural Network, FFNN) – это один из базовых и наиболее простых типов искусственных нейронных сетей. В такой сети информация передаётся строго в одном направлении: от входного слоя к выходному через один или несколько скрытых слоёв, без циклов и обратных связей [10].

Направление машинного обучения, которое использует многослойные нейронные сети называется глубоким обучением [2]. Модели глубокого

обучения способны эффективно решать более сложные задачи, например, задачи компьютерного зрения [11-27].

1.1.2 Принцип работы сверточных нейронных сетей

Понятие компьютерного зрения относится к области искусственного интеллекта, которая изучает методы и технологии обработки изображений и видео с целью понимания и интерпретации визуальной информации компьютерами [28]. Обычно для таких задач используется особый тип нейронных сетей – сверточные нейронные сети (или CNN, от англ. Convolutional Neural Networks) (Рисунок 4) [29]. Современные модели распознавания объектов в реальном времени, такие как YOLO и SSD, основанные на принципах сверточных нейронных сетей (CNN), будут описаны в разделе 2.2. В рамках данной главы рассмотрим базовые принципы функционирования CNN как одного из ключевых классов нейросетевых архитектур, применяемых в задачах компьютерного зрения.

В таких задачах изображение, как правило, представляется в виде трёхмерного массива (тензора), где каждый из трёх каналов (R, G, B) содержит матрицу чисел, отражающих уровень яркости соответствующего цвета в каждом пикселе. Таким образом, сверточная нейронная сеть, как и другие нейронные сети, обрабатывает и анализирует числа.

Типичная архитектура сверточной нейронной сети (CNN) включает следующие компоненты:

1. сверточные слои для извлечения признаков;
2. нелинейные функции активации (например, ReLU);
3. слои подвыборки (слои субдискретизации, пулинг) для снижения размерности;
4. операцию выравнивания (flatten);
5. полносвязные слои (Dense);

6. выходной слой, соответствующий задаче (например, softmax для классификации).

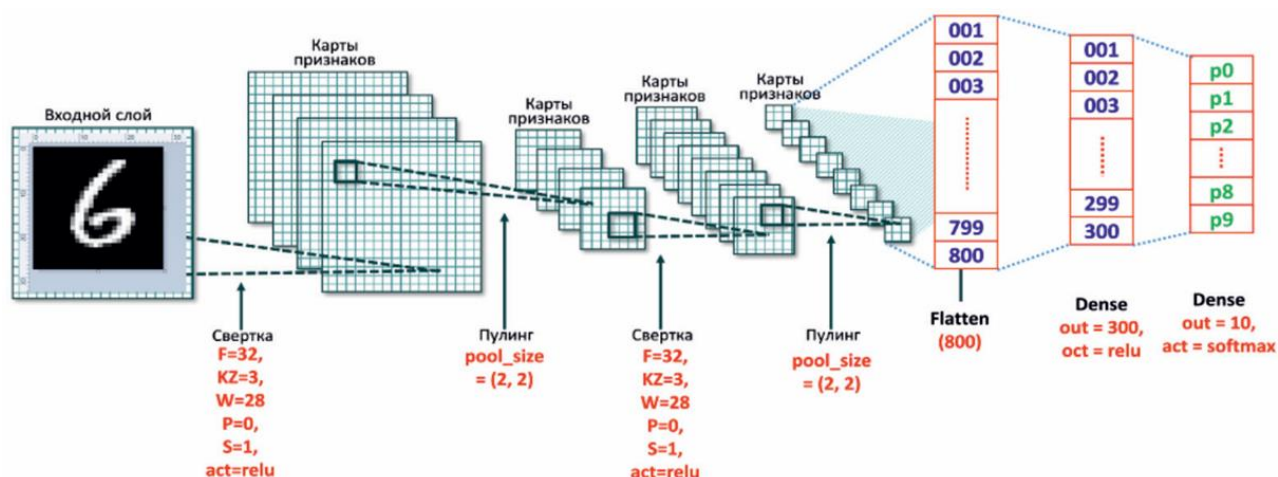


Рисунок 4 – Пример архитектуры сверточной нейронной сети (CNN) [15]

Сверточные слои и слои субдискретизации являются отличительной чертой CNN. Именно они вместе позволяют эффективно извлекать и обобщать признаки из входных данных, чаще всего изображений. Опишем принцип их работы.

Сверточный слой состоит из обучаемых фильтров, которые "сканируют" входное изображение с заданным шагом (Рисунок 5) [15]. Каждый фильтр F (например, 3×3 или 5×5) действует как матрица весов, применяемая к локальным областям изображения (матрицы I) – эта операция называется свёрткой. Результатом является карта признаков C , где каждая точка отражает степень активации фильтра.

Сверточный слой выполняет роль детектора признаков, выявляя элементы вроде границ, углов и текстур. Благодаря повторному применению фильтра по всему изображению, снижается количество параметров и повышается устойчивость к смещению объектов.

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

 \otimes

1	0	-1
1	0	-1
1	0	-1

 $=$

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

$I(8 \times 8)$
 $F(3 \times 3)$
 $C(4 \times 4)$

Рисунок 5 – Процесс свертки [15]

Свертка применяется к каждому слою входных данных: «... если изображение монохромное, матрица изображения I будет иметь один слой. Если изображение является цветным, матрица изображения I представлена тремя слоями матрицы, соответствующими трем цветам (R, G, B), отсюда матрица C также представлена тремя соответствующими слоями» [15] (Рисунок 6).

$6 \times 6 \times 3$
 $3 \times 3 \times 3$
 4×4

Рисунок 6 – Свертка каждого из слоев R, G, B цветного изображения [15]

При этом свёртка может уменьшать размер выходного тензора. Чтобы это контролировать, используют padding – добавление значений по краям изображения, чаще всего нулей (zero padding), что позволяет сохранить размеры или избежать потери информации по границе [15]. Также может использоваться заполнение константами, средними значениями или отражёнными пикселями.

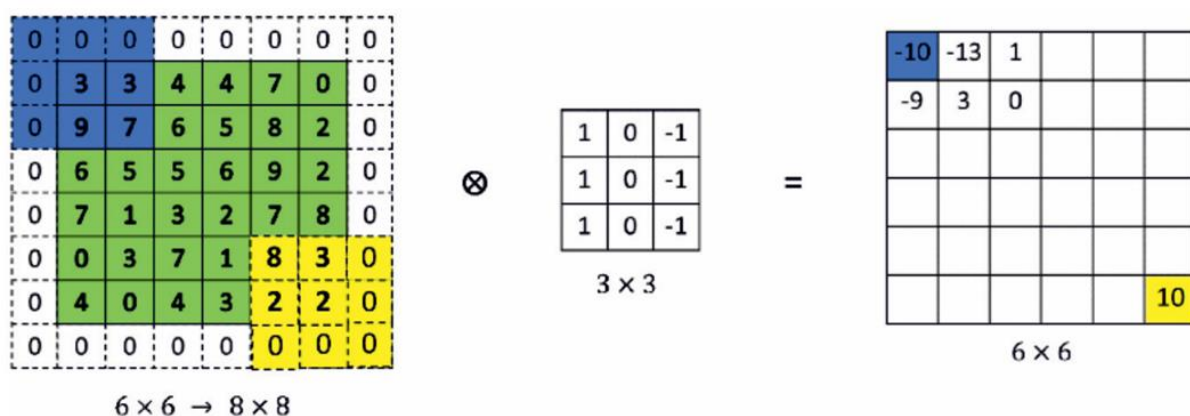


Рисунок 7 – Принцип работы padding (zero padding) [15]

После свёрточного слоя обычно следует слой субдискретизации [16], чаще в виде максимального пулинга Max Pooling. Он уменьшает пространственные размеры карты признаков, сохраняя важную информацию. Например, Max Pooling с окном 2x2 и шагом 2 выбирает максимум из каждого блока, формируя более компактное представление. Это снижает вычислительную нагрузку и повышает устойчивость к небольшим искажениям. Также может применяться средний пулинг (Average Pooling) – «для вычисления среднего среди всех значений анализируемой области», и пулинг суммы (Sum Pooling) – «для определения их суммы» [30] (Рисунок 8).

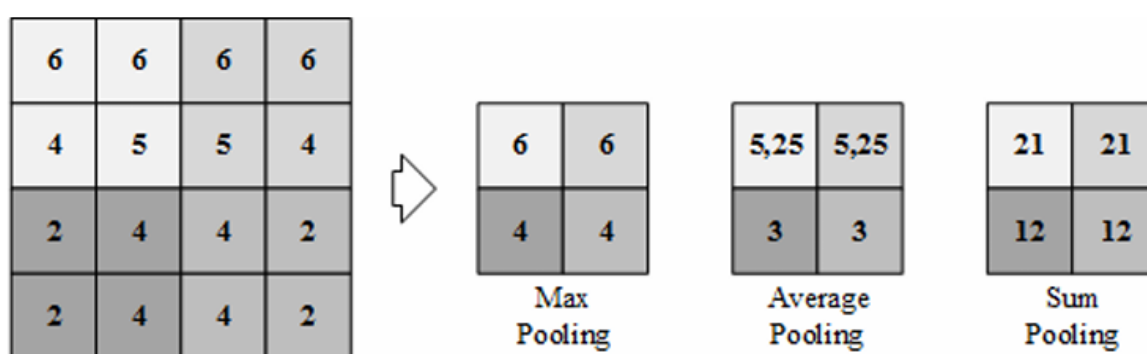


Рисунок 8 – Принцип работы максимального пулинга, среднего и пулинга суммы [30]

Кратко о роли других составных частей сверточной нейронной сети. Функция активации применяется после свертки. Она делает сеть нелинейной, позволяя ей учиться сложным функциям. После нескольких слоев свертки и пулинга данные "расплющиваются" в вектор (flatten). Этот вектор поступает в полносвязную (Dense) сеть, как в обычной нейронной сети. Эти слои называются слоями выравнивания. Последний из них – выходной слой, дает итоговый результат. В нем часто используется функция активации softmax (для многоклассовой классификации), который выдает вероятности принадлежности к классам [16], или sigmoid (для бинарных задач) – вероятность принадлежности к классу.

1.1.3 Принцип работы трансформеров

Помимо сверточных нейронных сетей, в последние годы широкое распространение получили архитектуры, основанные на механизме самовнимания (self-attention), известные как трансформеры. Изначально разработанные для задач обработки естественного языка [31-35], трансформеры () были успешно адаптированы к задачам компьютерного зрения [36-48] – в частности, классификации и детекции объектов (модели ViT, DETR и их производные). Некоторые из этих моделей были адаптированы для работы в реальном времени, например, RT-DETR, которая будет подробно рассмотрена в разделе 2.1. В рамках данной главы рассмотрим общие принципы работы трансформеров и механизм самовнимания, лежащий в их основе [49-51].

В таблице 1 представлены отличия CNN- и трансформерных подходов на уровне базовых механизмов, архитектур и моделей, что способствует более структурному пониманию их специфики.

Таблица 1 – Сравнение CNN- и трансформерных подходов в задачах компьютерного зрения

Уровень	CNN-подход	Трансформерный подход
Механизм	Свертка (convolution)	Самовнимание (self-attention)
Архитектура	CNN (AlexNet, ResNet...)	Transformer
Модели	YOLO, SSD и т.д.	ViT, DETR, RT-DETR и т.д.

Классическая архитектура Transformer состоит из повторяющихся блоков, включающих следующие ключевые компоненты:

1. Self-Attention (самовнимание) – каждый элемент входной последовательности (или патч изображения) взаимодействует с каждым другим, вычисляя вес их значимости. Это позволяет уловить глобальные зависимости (например, что объект на заднем плане влияет на интерпретацию объекта на переднем).

2. Multi-Head Attention – self-attention применяется параллельно с разными «головами» (набором обучаемых весов), что позволяет сети изучать разные аспекты внимания одновременно.

3. Position Encoding – так как трансформеры не имеют встроенного понимания порядка, как, скажем, RNN (рекуррентные нейронные сети), им добавляют позиционные признаки (например, синусоидальные или обучаемые вектора).

4. Feedforward слои – после блока внимания выход проходит через полносвязные слои с активацией.

5. Нормализация и остаточные связи (residual connections) – для стабилизации и ускорения обучения.

Self-Attention (самовнимание) – это ключевой компонент трансформерных архитектур, позволяющий модели учитывать взаимосвязи между всеми элементами входных данных. В контексте компьютерного зрения каждый входной элемент – это, как правило, вектор признаков, соответствующий

определённому участку изображения (патчу или пиксельному региону).
Механизм работает следующим образом:

1. Векторы Query, Key и Value. Для каждого входного вектора x_i сеть формирует три обучаемых проекции: Query (Q), Key (K) и Value (V). Эти проекции получаются умножением на разные обучаемые матрицы (формула 9):

$$Q_i = x_i W^Q, \quad K_i = x_i W^K, \quad V_i = x_i W^V \quad (9)$$

2. Вычисление внимания. Для каждого элемента Query Q_i рассчитывается его "сходство" (attention score) со всеми элементами Key K_j с помощью скалярного произведения (формула 10). Результат нормализуется через softmax, чтобы получить веса, отражающие важность других элементов относительно текущего (формула 11).

$$\text{Attention}(Q_j, k_i) = \frac{Q_i \cdot K_j^T}{\sqrt{d_k}}, \quad (10)$$

$$\alpha_{ij} = \text{softmax}_j = \left(\frac{Q_i \cdot K_j^T}{\sqrt{d_k}} \right) \quad (11)$$

где d_k – обозначает размерность векторов.

3. Взвешенное суммирование Value. Выходной вектор для x_i – это сумма всех Value V_j , взвешенных по α_{ij} (формула 12):

$$z_i = \sum_j \alpha_{ij} V_j, \quad (12)$$

где z_i – вектор, выход механизма самовнимания для i -го входного элемента (например, патча изображения).

Таким образом, каждый выходной вектор содержит информацию о том, какие другие части входа были для него наиболее значимы – это и есть "внимание". Благодаря этому трансформеры могут улавливать дальние и сложные зависимости в изображении, что особенно полезно в задачах, где контекст важен для распознавания объекта (например, различение пешехода и манекена по окружению).

Многоголовое внимание (Multi-Head Attention) является расширением базового self-attention. Его основная идея заключается в том, чтобы позволить модели одновременно учитывать различные аспекты взаимосвязей между элементами входных данных, тем самым повышая выразительную способность сети.

В отличие от обычного механизма самовнимания, в котором для каждого входного элемента формируются единственные векторы Query, Key и Value, в многоголовом внимании таких наборов создается несколько – по числу так называемых голов внимания. Каждая «голова» обучается независимо, с использованием своих собственных проекций параметров W_i^Q , W_i^K , W_i^V и способна фокусироваться на разных признаках входа. То есть для каждой головы i вычисляется свой набор attention-весов и соответствующих выходных векторов z_i , аналогично обычному self-attention.

При количестве голов h (от англ. heads) на выходе полученные вектора из разных голов z_1, z_2, \dots, z_h конкатенируются (объединяются по оси признаков) и далее проходят через линейное преобразование с обучаемой матрицей W^O . Таким образом, Multi-Head Attention реализует функцию (формула 13):

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h) W^O \quad (13)$$

где W^O – это матрица линейного преобразования, обучаемый параметр;
 $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$.

Такой подход позволяет каждой голове модели анализировать различные пространственные или семантические зависимости в изображении. Это особенно важно для задач детекции объектов, в которых одни признаки могут зависеть от формы, а другие – от взаимного расположения элементов.

Архитектура трансформеров полностью основана на механизме внимания, и сама по себе не учитывает порядок элементов во входной последовательности. Чтобы передать информацию о положении элементов, используется позиционное кодирование (Position Encoding) – векторы, добавляемые к эмбедингам токенов. В компьютерном зрении токенами обычно являются небольшие фрагменты изображения (патчи), которые после развёртывания и линейного преобразования становятся векторными представлениями, аналогично токенам слов в NLP.

Существует два основных подхода: синусоидальное позиционное кодирование и обучаемое позиционное кодирование. Синусоидальное позиционное кодирование или Fixed Positional Encoding было предложено в оригинальной статье «Attention is All You Need» [31]. Оно может быть выражено следующими формулами (формула 14, формула 15):

$$PE_{(pos, 2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \quad (14)$$

$$PE_{(pos, 2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right), \quad (15)$$

где PE – Positional Encoding (синусоидальный позиционный вектор);

pos – позиция токена в последовательности;

i – индекс размерности;

d_{model} – размерность скрытого пространства (т.е. позиционного вектора и токена).

Такое кодирование не содержит обучаемых параметров: позиционные эмбединги вычисляются по фиксированной формуле и могут быть заданы для любого значения pos , даже за пределами обучающей выборки. Благодаря периодическим и непрерывным функциям синуса и косинуса кодирование охватывает широкий спектр частот, позволяя модели учитывать как локальные, так и глобальные зависимости. Однако жёсткая структура таких кодировок ограничивает их адаптацию к конкретной задаче.

В подходе обучаемого позиционного кодирования (Learned Positional Embedding) каждая позиция в пределах фиксированной длины имеет свой обучаемый вектор, что делает ее более гибкой, позволяя модели выучить наилучшее представление порядка (формула 16). Однако модель становится плохо масштабируемой за пределы длины, использованной в обучении.

$$PE_{pos} \in \mathbb{R}^{d_{model}}, \quad (16)$$

где PE – Positional Embedding (обучаемый позиционный вектор)

$\mathbb{R}^{d_{model}}$ – векторное пространство размерности d_{model} , то есть множество всех вещественных векторов длины d_{model} .

Подведем итог того, как работает трансформер. Каждый входной элемент сначала преобразуется в эмбединг, к которому добавляется позиционное кодирование. Затем последовательность проходит через несколько одинаковых блоков (Transformer Blocks), каждый из которых состоит из многоголового внимания, нормализации, остаточной связи и позиционно-независимого полносвязного слоя. На каждом этапе модель последовательно уточняет представления токенов с учётом их контекста в последовательности. На выходе из трансформера получается улучшенное представление для каждого токена, которое учитывает как локальные, так и глобальные зависимости. Эти представления могут быть использованы для дальнейших задач, в том числе и

для классификации: в таком случае выходные представления каждого токена объединяют в один агрегированный вектор, который затем передаётся в полносвязный слой для предсказания категории.

В разделе 1.1 представлены базовые принципы работы нейронных сетей – от устройства искусственного нейрона до ключевых архитектурных подходов. Рассмотрены сверточные нейронные сети (CNN), остающиеся основным инструментом в задачах компьютерного зрения, а также трансформеры – более современные модели, способные учитывать глобальный контекст. Проведен сравнительный обзор их архитектурных особенностей, преимуществ и ограничений в контексте задач детекции объектов.

1.2 Методы обучения нейронных сетей и обработки данных

Обучение нейросетевой модели представляет собой процесс настройки её внутренних параметров (весов) с целью минимизации ошибки при решении поставленной задачи. Наиболее распространённый подход основан на градиентной оптимизации: параметры модели обновляются в направлении антиградиента функции потерь. Для эффективного вычисления градиентов используется метод обратного распространения ошибки (backpropagation), реализующий правило цепного дифференцирования и позволяющий обучать глубокие нейросети за счёт поэтапного распространения сигнала от выхода к входу [52].

В зависимости от наличия и структуры разметки, обучение нейросетевых моделей может быть с учителем, без учителя, с частичной разметкой (semi-supervised) или с подкреплением [53]. В задачах распознавания объектов традиционно используется обучение с учителем, при котором модель обучается на размеченных изображениях, содержащих координаты ограничивающих рамок и метки классов.

Процесс обновления параметров реализуется с помощью алгоритмов оптимизации. Наиболее популярны методы, основанные на градиентном спуске, такие как SGD (Stochastic Gradient Descent), Adam, RMSprop и Adagrad. Они различаются стратегиями выбора шага обучения, а также тем, учитывают ли историю градиентов. Например, Adam использует экспоненциальное сглаживание градиентов и их квадратов, что ускоряет сходимость на шумных и разреженных данных [54, 55]. Другой распространенный метод – SGD (Stochastic Gradient Descent), который обновляет веса на основе градиентов от случайных подвыборок. Несмотря на простоту, SGD часто обеспечивает лучшую обобщающую способность, особенно в случае больших датасетов и при наличии тщательной настройки скорости обучения [56]. Выбор метода зависит от задачи: Adam предпочтителен для быстрого прототипирования и нестабильных градиентов, а SGD – для финального обучения на крупных наборах данных, когда важна точность модели.

Несмотря на доминирование градиентных методов, существуют и альтернативные подходы. К ним относятся эволюционные алгоритмы, в частности генетические. [57]. Также применяются методы второго порядка (например, алгоритм Ньютона, L-BFGS), учитывающие кривизну функции потерь. Хотя они могут обеспечить более быструю сходимость, на практике такие методы редко используются из-за высокой вычислительной нагрузки [58]. В задачах обучения с подкреплением также применяются как градиентные, так и неградиентные методы, направленные на оптимизацию стратегии поведения [59]. Отдельное направление составляют автоматизированные методы поиска архитектур (AutoML, NAS), при которых подбор структуры сети сочетается с её обучением [60]. Однако в задачах распознавания объектов в реальном времени такие подходы пока применяются ограниченно из-за вычислительной сложности.

В прикладных задачах компьютерного зрения, включая распознавание объектов, часто используются предобученные модели – нейросети, заранее

обученные на крупных универсальных наборах изображений, например, таких как COCO [61]. Это позволяет сократить затраты на обучение и повысить точность за счёт уже сформированных признаков нижнего уровня [62].

Дообучение (fine-tuning) предобученных моделей является частным случаем переноса обучения (transfer learning). Оно используется для адаптации предобученной модели к специфическим условиям применения (домену). Под доменом здесь понимается совокупность характеристик данных, включая тип сцены, погодные условия, инфраструктуру и т.п. Это особенно актуально при переходе от общего набора изображений к специализированному датасету, содержащему дорожные объекты в различных погодных условиях или инфраструктурных контекстах. В зависимости от объёма выборки и степени отличия нового домена, возможен как частичный пересчёт верхних слоёв модели, так и полное обновление всех параметров с использованием пониженного значения шага обучения (learning rate) [63].

При дообучении на ограниченных выборках возрастает риск переобучения – ситуации, когда модель хорошо воспроизводит обучающие данные, но теряет способность к обобщению на новых примерах. Это происходит, когда модель чрезмерно подстраивается под шум или частные особенности обучающей выборки, в результате чего её точность на тестовых данных снижается [52]. Для снижения этого риска применяются методы регуляризации [52], ранней остановки (early stopping) [64], аугментации данных [65], а также использование пониженного значения learning rate [66]. Кроме того, при наличии небольшого количества специализированных данных рекомендуется замораживать нижние слои модели, сохраняя предварительно обученные признаки [67].

В задачах распознавания объектов на видео модели (такие как YOLO (все версии), SSD, RT-DETR, Faster R-CNN, DETR и его производные, YOLO-World, GroundingDINO, DINOv2 и др.) обрабатывают каждый кадр независимо, без учета временного контекста [68]. Это означает, что обучение и инференс происходят на уровне отдельных изображений, без использования межкадровых

связей или трекинга [69]. Такой подход широко используется, когда требуется лишь детекция объектов, а не их отслеживание во времени (как в задачах отслеживания объектов, распознавания действия, предсказаний траектории движения объекта [70]). Существуют и альтернативные подходы, где временной контекст все же используется [71], даже если нет задачи трекинга и т.п.: такие модели применяются реже, в основном, когда важно учитывать динамику сцены (например, при плохом освещении, быстрых объектах или видеοшуме). В данной работе их применение не рассматриваются.

Перед подачей изображений в нейросетевую модель обычно проводится масштабирование и нормализация входных данных для обеспечения стабильности и эффективности обучения.

Масштабирование (или *resize*) – это изменение размеров изображений до фиксированного разрешения для подачи в нейросетевую модель [52]. При обучении оно обеспечивает одинаковый размер входных данных, облегчая пакетную обработку и ускоряя обучение. При инференсе масштабирование позволяет привести входное изображение к тому размеру, на котором обучалась модель, обеспечивая корректную работу свёрток и других слоёв.

Нормализация уменьшает разброс значений пикселей. При обучении нормализация (например, приведение значений пикселей к диапазону $[0, 1]$ или стандартизация) помогает ускорить сходимость, стабилизирует градиентный спуск и снижает переобучение. [52]. При инференсе необходимо использовать тот же тип нормализации, что и во время обучения (особенно если модель предобучена). Иначе распределение входных данных будет отличаться от того, к которому адаптировалась модель, что резко ухудшает точность.

При использовании специализированных фреймворков для обучения моделей важно учитывать их функциональные особенности. Например, во фреймворке *Ultralytics* [72] большинство преобразований, таких как флипы, повороты и масштабирование с элементами случайности, выполняются автоматически как часть встроенной аугментации. В моделях, реализованных

через библиотеку Hugging Face Transformers [73], такие преобразования по умолчанию не применяются и должны быть явно заданы в коде предобработки.

Проблема дисбалансировки классов является одной из ключевых задач в области машинного обучения и компьютерного зрения, особенно в задачах детекции объектов. Она возникает, когда в обучающем наборе данных одни классы представлены значительно большим количеством примеров по сравнению с другими, что приводит к снижению точности распознавания менее распространённых категорий и смещению модели в пользу доминирующих классов [74, 75]. Основные подходы к решению данной проблемы включают методы балансировки данных, такие как oversampling и undersampling, а также алгоритмические решения – модификацию функции потерь с использованием весов классов и применение специальных loss-функций, например focal loss [76, 77]. Кроме того, современные методы обучения предлагают стратегическое формирование батчей (stratified sampling) и использование аугментаций, направленных на увеличение представительности редких классов [78].

Для обеспечения качества обучения и стабильности работы модели данные необходимо предварительно проверять на корректность: устранять пустые или повреждённые файлы, пропущенные значения, а также убедиться в соответствии форматов изображений и аннотаций установленным требованиям.

В данном подразделе обобщённо рассмотрены ключевые методы обучения нейронных сетей и подготовки данных, а также основные подходы к обеспечению стабильности и эффективности процесса обучения в задачах распознавания объектов.

1.3 Методы оценки моделей распознавания объектов в реальном времени

Показатели качества работы нейросетевой модели в задаче распознавания объектов в реальном времени делятся на две группы: показатели точности и производительности.

Начнем с рассмотрения показателей точности. В отличие от задач простой классификации, распознавание (детекция) объектов требует не только оценки качества классификации, но также учёта пространственного положения обнаруженных объектов, что влечёт за собой использование специализированных метрик [79].

Одной из основных таких метрик является Intersection over Union (IoU) – коэффициент пересечения предсказанной и истинной ограничивающих рамок. IoU определяется как отношение площади пересечения к площади объединения двух рамок (Рисунок 11):

Чем выше значение IoU, тем точнее модель определила положение объекта. На практике часто используется пороговое значение $\text{IoU} \geq 0.5$ или 0.75 , при котором предсказание считается корректным (true positive) [80].

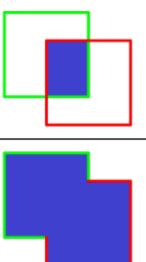
$$\text{IoU} = \frac{\text{area of overlap}}{\text{area of union}} = \frac{\text{area of intersection}}{\text{area of union}}$$


Рисунок 11 – Демонстрация принципа работы показателя IoU [83]

Для определения качества классификации обнаруженных объектов применяются следующие метрики [81]:

Precision (точность) – доля корректно предсказанных объектов от общего количества предсказанных (формула 17):

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (17)$$

Recall (полнота) – доля корректно предсказанных объектов от общего количества истинных объектов (формула 18):

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (18)$$

F1-мера – гармоническое среднее между точностью и полнотой (формула 19):

$$\text{F1} = 2 \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (19)$$

где TP – число истинно положительных срабатываний;

FP – ложноположительные;

FN – ложноотрицательные.

Наиболее широко используемой метрикой в задачах обнаружения объектов (object detection) является Average Precision (AP) – интегральная оценка, характеризующая качество модели как площадь под кривой precision-recall. Она рассчитывается при различных порогах уверенности (confidence thresholds) и отражает зависимость полноты (recall) от точности (precision). (Рисунок 12). Уверенность – это вероятность, с которой модель считает, что на изображении действительно присутствует объект определённого класса. Она вычисляется как произведение вероятности наличия объекта в рамке и вероятности, что он принадлежит определённому классу. В задачах object detection AP, как правило, вычисляется при фиксированном пороге пересечения предсказанной и истинной области (IoU).

Для многоклассовых задач рассчитывается mean Average Precision (mAP) как среднее значение AP по всем классам [82]. Например:

- mAP@0.5 – AP при пороге IoU = 0.5;
- mAP@[0.5:0.95] – среднее значение AP при IoU от 0.5 до 0.95 с шагом 0.05 (используется в COCO evaluation) [83].

Метрика mAP является де-факто стандартом для оценки моделей в соревнованиях и репозиториях, таких как COCO, Pascal VOC, Open Images и др. [84]. Также она является критически важным показателем качества моделей распознавания объектов для практического применения в задачах помощи водителю и особенно автономного управления.

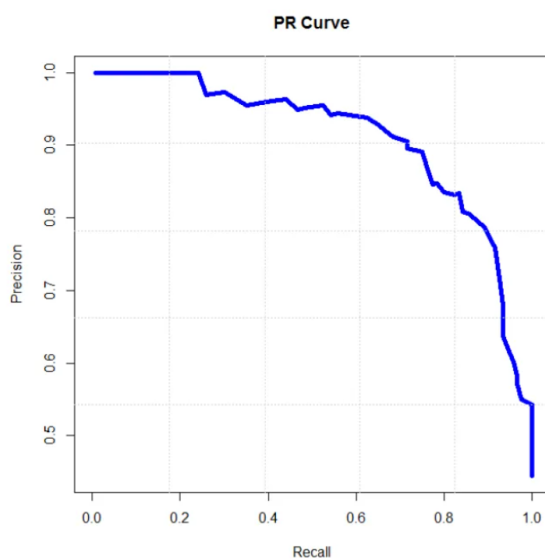


Рисунок 12 – Пример кривой PR – Precision-Recall

Наиболее распространённой метрикой остаётся mAP@0.5 – средняя точность при пороге перекрытия (IoU) не менее 0.5. В большинстве исследовательских и прикладных решений значение $\text{mAP}@0.5 \geq 0.5$ считается минимально приемлемым уровнем точности [80, 85]. В более ответственных задачах, например, системах автономного вождения, рекомендуется достигать значений не ниже 0.7-0.8 [86].

Для более строгой оценки качества часто используется метрика $mAP@[0.5:0.95]$, при которой итоговая точность усредняется по нескольким порогам IoU (от 0.5 до 0.95 с шагом 0.05) [82]. В рамках исследовательских бенчмарков, таких как COCO или PASCAL VOC, значение $mAP@[0.5:0.95] \approx 0.4$ принято считать удовлетворительным [82]. Однако в реальных условиях автономного вождения этого уровня может быть недостаточно, особенно для критически важных классов объектов (например, пешеходы или велосипедисты), где ошибки недопустимы. В промышленной практике, включая вызовы Waymo Open Dataset Challenge и nuScenes, используются более строгие метрики, такие как weighted mAP, mAPH и вспомогательные показатели устойчивости детекции во времени [87].

Таким образом, выбор метрики и оценка допустимого уровня точности зависят от задач и контекста применения модели. В данной работе основной акцент сделан на достижении $mAP@0.5$ не ниже 0.5, как ориентир для практической применимости в системах помощи водителю.

Для задач распознавания объектов на дороге в режиме реального времени, особенно в системах автономного управления транспортом, важно учитывать не только точность модели, но и её производительность. К ключевым метрикам относятся: время инференса (latency), количество обрабатываемых кадров в секунду (FPS), загрузка вычислительных ресурсов и размер модели [52, 85].

Время инференса (latency) отражает, сколько миллисекунд требуется модели для обработки одного изображения. Для систем, работающих в условиях движения, допустимым порогом считается задержка не выше 66 мс (что соответствует 15 FPS) [88].

FPS (Frames Per Second) указывает на способность модели обрабатывать поток изображений с нужной скоростью. В задачах автономного вождения нижней границей считается $FPS \geq 15$, однако более безопасными считаются значения 20–30 FPS и выше, особенно при высокой скорости движения [88, 89].

Использование ресурсов (GPU/CPU Load, Memory Usage) должно быть адаптировано под конкретную платформу. Например, на edge-устройствах (NVIDIA Jetson, Qualcomm Snapdragon Ride и др.) допустимое потребление GPU-памяти ограничено 1–2 ГБ, а загрузка GPU не должна блокировать параллельные процессы (включая SLAM, трекинг, планирование) [90].

Вес модели (Model Size) определяет возможность её размещения в памяти устройства и загрузки в ограниченные среды. На практике модель объемом до 100 МБ может быть использована на edge-устройствах, а более тяжелые (>500 МБ) – только при наличии полноценного GPU [89].

Таким образом, в контексте беспилотных автомобилей, под режимом «реального времени» подразумевается способность модели:

1. обеспечивать $FPS \geq 15-30$;
2. иметь $latency \leq 66$ мс;
3. потреблять ограниченный объем ресурсов (в зависимости от платформы);
4. быть достаточно компактной для интеграции в целевое устройство.

Баланс между точностью и производительностью является важным критерием при выборе модели для внедрения в практических системах, особенно в условиях ограниченных вычислительных ресурсов.

Кроме количественных метрик, в процессе валидации широко применяется визуализация результатов, позволяющая выявить систематические ошибки модели: пропущенные объекты, ложные срабатывания, смещения ограничивающих рамок и др. Такой анализ особенно важен при дообучении моделей на специфических доменах, когда стандартные метрики не отражают полной картины [91].

Таким образом, в задаче детекции объектов в реальном времени ключевыми критериями оценки моделей являются точность (mAP) и производительность (время инференса, FPS, использование ресурсов и размер модели). В данном подразделе были рассмотрены основные метрики и

сформулированы минимальные требования к ним с учетом специфики задач автономного управления транспортом.

1.4 Литературный обзор работ по распознаванию объектов на дороге с применением нейросетей

Ряд зарубежных исследований посвящён сравнительному анализу современных архитектур распознавания объектов, применяемых в задачах автономного вождения. В работе «Comparative analysis of object detection algorithms for autonomous vehicle applications» рассматриваются и экспериментально сравниваются модели YOLOv4, SSD, RetinaNet, Mask R-CNN и R-FCN на наборе данных, имитирующем дорожные сцены [92]. Оценка производилась по метрикам точности (mAP) и скорости (FPS). Авторы подтверждают, что одностадийные модели, такие как YOLOv4 и SSD, показывают высокую производительность по скорости обработки, что делает их предпочтительными для задач, требующих работы в реальном времени. В то же время двухстадийные модели, включая Mask R-CNN и R-FCN, демонстрируют более высокую точность, особенно при обнаружении мелких и частично перекрытых объектов, что критично для безопасной навигации в сложных условиях. Таким образом, выбор модели определяется необходимым балансом между скоростью и точностью, что особенно важно при проектировании систем автономного вождения, где критично как быстрое реагирование, так и надёжное распознавание объектов.

В статье «Deep Learning approaches for Object Detection in Self-Driving Cars» рассматриваются как международные, так и российские исследования, акцентируется внимание на практических аспектах внедрения таких технологий в реальные условия дорожного движения [93]. Особое внимание уделяется методам transfer learning и fine-tuning, которые позволяют использовать предобученные модели (например, на COCO или ImageNet) и адаптировать их к

локальным условиям, включая особенности дорожной инфраструктуры, нестандартные погодные условия или поведение других участников движения. Авторы подчёркивают, что такие методы позволяют сократить время обучения и повысить устойчивость моделей к вариативности входных данных, что критически важно для систем автономного управления в реальном мире.

Среди применяемых CNN-моделей, применяемых к рассматриваемой задаче зарубежными исследователями, часто встречается модель YOLO. В одной из таких работ представлена модель MST-YOLO, разработанная для эффективного обнаружения мелких объектов в условиях автономного вождения [94]. Модель улучшает точность обнаружения мелких объектов, таких как пешеходы и дорожные знаки, за счёт оптимизации архитектуры YOLO и внедрения многоуровневого масштабного обучения. Эксперименты на датасетах KITTI и COCO показали повышение точности при сохранении высокой скорости обработки. В еще одной работе те же авторы предлагают улучшенную версию YOLOv5, оптимизированную для работы с изображениями, полученными с камер, установленных на транспортных средствах [95]. Модель демонстрирует высокую точность и скорость обнаружения объектов в реальном времени, что делает её пригодной для применения в системах автономного вождения.

Применение трансформеров к заявленной задаче также активно исследуется зарубежом. Например, в статье «Automatic Vehicle Detection using DETR» 2025 года авторы исследуют применение модели DETR для автоматического обнаружения транспортных средств в сложных дорожных условиях [96]. Предложен метод Co-DETR, улучшающий обучение и механизмы внимания в DETR. Эксперименты на датасете BadODD показывают, что предложенный подход превосходит традиционные методы, такие как YOLO и Faster R-CNN, по точности и эффективности, особенно в условиях плохой видимости и сложной дорожной обстановки.

Рассмотрим и другие модификации модели DETR, применяемые зарубежными авторами в задаче дорожной детекции в режиме реального

времени. RT-DETRv3 представляет собой улучшенную версию модели RT-DETR, предназначенную для обнаружения объектов в реальном времени [97]. Модель внедряет иерархическую плотную положительную супервизию и стратегию обучения с возмущением самовнимания, что позволяет повысить точность обнаружения без увеличения задержки обработки. Эксперименты на датасете COCO показали превосходство RT-DETRv3 над предыдущими версиями и моделями серии YOLO по точности при сохранении высокой скорости обработки.

LW-DETR – это лёгкая модель на основе трансформеров, предназначенная для замены YOLO в задачах обнаружения объектов в реальном времени [98]. Модель использует стек ViT-энкодера, проектор и неглубокий DETR-декодер, а также внедряет техники, такие как межоконное и глобальное внимание, для снижения сложности энкодера. Эксперименты показали, что LW-DETR превосходит существующие детекторы в реальном времени, включая YOLO и его варианты, по точности на датасетах COCO и других.

В еще одной работе представлена улучшенная модель обнаружения объектов на основе DETR, предназначенная для автономного вождения [99]. Модель внедряет многомасштабное извлечение признаков и механизм группового аксиального внимания, что позволяет повысить точность и скорость вывода. Эксперименты на датасетах COCO, PASCAL VOC и KITTI показали улучшение средней точности на 3.3%, 4.5% и 3% соответственно, а также увеличение FPS на 84% по сравнению с базовой моделью DETR.

В ряде зарубежных работ также были предложены гибридные модели, совмещающие CNN и Transformers [100-102]. Например, в одной из этих работ представлена гибридная модель STAFFNet, объединяющая CNN и трансформеры для эффективного распознавания объектов в сложных дорожных условиях [100]. Данная модель демонстрирует высокую точность на датасете KITTI: 95.37% для простых, 93.64% для умеренных и 85.49% для сложных условий. STAFFNet превосходит другие современные методы, включая DETR и

YOLOv5, особенно в задачах обнаружения мелких и удалённых объектов. Модель также обеспечивает высокую скорость обработки, что делает её пригодной для реального времени.

Возможность применения глубоких нейронных сетей к задаче распознавания дорожных объектов изучалась и отечественными исследователями. В статье «Анализ и обучение модели нейронной сети распознавания дорожных знаков» авторы предлагают собственную модель, основанную на архитектуре YOLOv7, модификация которой была направлена на повышение точности и скорости распознавания [29]. Модель была дообучена на российском датасете RTSD, содержащем 146 классов дорожных знаков по ГОСТ Р 52290-2004. Результаты показали точность предсказаний 0,847 и полноту 0,811, что подтверждает эффективность предложенного подхода для задач автономного вождения.

Одним из российских примеров применения сверточных нейронных сетей в задачах управления беспилотными автомобилями является проект команды Bauman Racing Team (МГТУ им. Баумана) [103]. Для распознавания объектов на трассе были использованы две нейросетевые модели: YOLOv5, предназначенная для детекции и классификации дорожных конусов по цвету, и модель KeyPoint, которая определяет пространственные характеристики объектов с помощью ключевых точек. Обе модели обучались на датасете FSOCO с более чем 20 000 изображений, охватывающим разнообразные погодные и географические условия. Обучение проводилось в облаке с использованием инфраструктуры Yandex.Cloud, что дало возможность быстро масштабировать вычислительные мощности. Этот кейс иллюстрирует успешное применение CNN в реальном времени на практике, а также демонстрирует важность вычислительных ресурсов и гибкой инфраструктуры при разработке систем автономного вождения.

В работе Минниханова Р.Н., Скибина В., Талипова Н.Г. и Катасёва А.С. [104] рассматривается задача распознавания сигналов светофора с использованием моделей глубокого обучения на основе сверточных нейронных

сетей. Для решения поставленной задачи были реализованы и сравнены две архитектуры: двухэтапная Faster R-CNN и одностадийная YOLOv8n. Обе модели были обучены на наборе данных LISA Traffic Light Dataset, дополненном изображениями, полученными из сети Интернет. Данные были предварительно размечены, а также дополнительно аугментированы для повышения обобщающей способности моделей. Обучение производилось в среде Python с использованием PyTorch на видеокарте RTX 2060 Super.

Сравнение моделей проводилось по метрике F1-score, скорости обработки (FPS) и размеру модели. Модель Faster R-CNN показала более высокую точность ($F1 = 0,907$), однако YOLOv8n лишь незначительно уступила по этому показателю ($F1 = 0,905$), при этом продемонстрировала существенно более высокую производительность – 55 кадров в секунду против 10 у Faster R-CNN – и в 27 раз меньший размер (5,94 МБ против 158 МБ). Таким образом, авторы приходят к выводу, что несмотря на немного меньшую точность, модель YOLOv8n благодаря своей компактности и высокой скорости обработки является предпочтительным решением для задач детекции светофоров в интеллектуальных транспортных системах.

В еще одной статье рассматривается применение моделей глубокого обучения для задачи обнаружения пешеходов в системах интеллектуальной поддержки водителей [105]. В работе проведён обзор методов обнаружения объектов на изображениях: традиционных (например, метод Виолы–Джонса, HOG, «набор слов») и основанных на глубоких нейросетях. Отмечено, что хотя традиционные методы менее ресурсоёмкие, они существенно уступают по точности современным нейросетевым архитектурам. Среди последних рассматриваются двухэтапные (R-CNN, Fast R-CNN, Faster R-CNN) и одноэтапные модели (YOLO, SSD), причём особое внимание уделено их применимости в системах реального времени.

Для оценки эффективности модели была реализована демонстрационная программа на Python с использованием OpenCV и Tkinter, функционирующая на

обычном ноутбуке без выделенного GPU. Эксперименты проводились на видеоданных с автомобильных регистраторов, взятых из открытых источников. Обнаружение пешеходов производилось при различных значениях параметра IoU, определяющего точность сопоставления предсказанных и истинных ограничивающих рамок. Оптимальный результат был достигнут при $\text{IoU} = 0.7$, при котором наблюдались высокая полнота детекции и минимальное количество ложноположительных срабатываний. Время обработки одного кадра при разрешении 1920×1080 составило около 0,15 секунд. Учитывая компромисс между точностью и скоростью, авторы пришли к выводу, что оптимальным выбором является использование модели YOLOv4-tiny, обладающей высокой производительностью при низких требованиях к вычислительным ресурсам.

В данном подразделе представлен обзор работ, посвящённых применению нейросетевых моделей глубокого обучения к задаче распознавания объектов на дороге. В зарубежной литературе широко представлены исследования, использующие как сверточные нейронные сети (CNN), так и архитектуры на основе трансформеров, а также гибридные подходы. К числу активно применяемых моделей относятся YOLO (включая версии YOLOv4, YOLOv5, YOLOv7, YOLOv8, YOLOv4-tiny, YOLOR), SSD, RetinaNet, Mask R-CNN и Faster R-CNN – со стороны CNN-подходов, а также DETR, RT-DETR, LW-DETR, Swin Transformer и STAFFNet – со стороны трансформеров и гибридных архитектур.

Анализ отечественных исследований показывает, что в работах российских авторов преобладает использование сверточных нейронных сетей. В частности, активно применяются модели YOLOv4, YOLOv4-tiny, YOLOv5 и YOLOv7, в том числе в задачах распознавания дорожных знаков, светофоров и пешеходов. Среди рассмотренных примеров можно выделить работы по распознаванию светофоров с использованием YOLOv7, распознаванию дорожных знаков с YOLOv5, а также детекции пешеходов на базе YOLOv4-tiny. В то же время трансформеры в российской литературе на момент подготовки обзора практически не применялись к задаче распознавания дорожных объектов

в реальном времени. Это может быть обусловлено несколькими факторами, включая их сравнительно недавнее появление в данной области (с 2023 года), высокие вычислительные требования, а также тем, что они пока не успели получить широкого распространения в отечественных исследованиях и прикладных проектах.

Выводы по первой главе

В первой главе изложены теоретические основы, необходимые для понимания принципов работы современных нейросетевых методов, применяемых в задачах распознавания объектов. Последовательно рассмотрены ключевые понятия – от устройства искусственного нейрона до ключевых архитектурных подходов и устройства конкретных современных архитектур для задач распознавания объектов, включая сверточные нейронные сети (CNN) и трансформеры.

Также описаны основные методы предобработки данных и обучения нейросетей, включая дообучение предобученных моделей и подходы к стабилизации процесса обучения. В разделе, посвящённом оценке моделей, не только представлены метрики качества, но и сформулированы минимальные требования к точности и производительности моделей с учётом специфики применения в реальном времени для автономного управления транспортом.

Завершающая часть главы содержит обзор актуальных отечественных и зарубежных исследований (с 2023 по 2025 год), что позволило выявить тенденции в выборе архитектур. YOLO – это проверенный и широко распространённый подход к детекции объектов, успешно применяемый в задачах компьютерного зрения как в России, так и зарубежом. Трансформеры же начали активно применяться в задаче дорожной детекции в реальном времени сравнительно недавно – с 2023 года. При этом в российской литературе по данной задаче на момент подготовки обзора трансформеры практически не

используются, что, вероятно, связано с высокой вычислительной сложностью таких моделей и их относительной новизной.

2 Модели, данные и инструменты для детекции дорожных объектов в реальном времени

2.1 Обзор предобученных моделей детекции объектов в реальном времени

2.2.1 Архитектура и особенности моделей YOLO

Существует множество различных предварительно обученных моделей. Выбор конкретной модели может зависеть от условий её применения. Предварительно обученные модели машинного обучения представляют собой модели, которые были обучены на больших наборах данных для выполнения конкретной задачи и затем доступны для использования другими пользователями без необходимости повторного обучения. Эти модели являются мощным инструментом в мире машинного обучения, так как они позволяют быстро и эффективно применять готовые решения к различным задачам.

Кроме того, такие модели могут быть дообучены на специализированных наборах данных, что позволяет адаптировать их под конкретную прикладную задачу или особенности целевого домена. Именно такой подход применяется в данной работе. Это требует взвешенного выбора не только данных для дообучения, но и модели-основы, способной обеспечить нужный баланс между точностью и производительностью в условиях задачи распознавания объектов на дороге в реальном времени.

Одноступенчатые модели, такие как YOLO (You Only Look Once), являются оптимальным выбором для задач детекции объектов в реальном времени, особенно в контексте автономного вождения. Их ключевое преимущество заключается в высокой скорости обработки: они выполняют детекцию за один проход по изображению, без необходимости в отдельном этапе генерации регионов интереса, как это происходит в двухступенчатых моделях

(например, Faster R-CNN). Это позволяет достигать высокой частоты кадров (FPS), что критично для своевременного реагирования на изменения в дорожной обстановке.

Модель YOLO была впервые представлена в 2015 году Джозефом Редмоном и его коллегами [91]. Модели семейства YOLO (You Only Look Once) являются первыми одноступенчатыми сверточными нейронными сетями (CNN), которые предназначены для обнаружения объектов в реальном времени. Главная идея – выполнить детекцию объектов за один проход через модель (в отличие от двухступенчатых подходов вроде R-CNN, где сначала генерируются регионы интереса, а потом классифицируются).

Принцип работы этих моделей заключается в следующем. YOLO разбивает входное изображение на сетку (например, 13×13 , 19×19), и каждая ячейка сетки отвечает за предсказание: наличия объекта, координат ограничивающего прямоугольника (bounding box), уверенности (confidence), класса объекта. Для каждого изображения модель предсказывает сразу все bounding box'ы и классы объектов, обрабатывая его как регрессионную задачу от изображения к координатам и меткам классов.

Общая структура моделей YOLO состоит из трех компонент: Backbone, Neck, Head. Входное изображение масштабируется и подаётся на backbone, где происходит извлечение признаков. Полученные карты признаков (feature maps) обрабатываются neck'ом, который объединяет информацию с разных уровней для улучшения детекции объектов различных размеров. Наконец, head производит предсказания ограничивающих рамок, классов и уровней уверенности – всё это выполняется за один проход через сеть.

Backbone (сеть-признакодобытчик) – это сверточная нейросеть, предназначенная для извлечения признаков из входного изображения. Она заменяет традиционные методы выделения признаков.

Neck (сеть объединения признаков) повышает качество детекции, обрабатывая признаки с разных уровней (scale-aware detection). Обычно

включает в себя FPN (Feature Pyramid Network) и PANet (Path Aggregation Network). Эти блоки позволяют использовать информацию с разных уровней глубины сети (разные масштабы объектов).

Head (голова для предсказания) – выходной блок, который предсказывает: координаты ограничивающих рамок (bounding boxes), вероятность наличия объекта (objectness score), принадлежность к классам (class scores). Предсказания производятся в несколько якорных точек на разных масштабах.

С момента появления YOLOv1 (2016) модель прошла через несколько итераций, каждая из которых вносила улучшения в точность и производительность [72]. YOLOv8 (2022-2023) – последняя на данный момент версия, предлагающая улучшенную точность и поддержку новых функций, таких как сегментация и отслеживание объектов.

Для повышения точности при обнаружении объектов различных форм и размеров YOLO использует механизм якорных рамок (anchor boxes) – заранее заданных шаблонов ограничивающих рамок разных пропорций. Вместо того чтобы предсказывать рамки «с нуля», модель лишь корректирует параметры этих якорей, приближая их к фактическому положению объекта на изображении. Это позволяет существенно улучшить результативность, особенно при наличии объектов разных масштабов в одной сцене.

Обучение YOLO происходит на основе комплексной функции потерь, которая включает в себя три компонента: ошибку локализации (насколько точно рамка описывает объект), ошибку классификации (насколько точно определён класс) и ошибку уверенности (насколько модель уверена в наличии объекта в рамке). Эта интеграция всех аспектов предсказания в одну функцию потерь позволяет YOLO обучаться эффективно и сбалансировано.

После получения предсказаний применяется процедура постобработки, наиболее важной частью которой является подавление немаксимумов (non-maximum suppression, NMS). Эта процедура устраняет избыточные рамки, которые перекрываются и «претендуют» на один и тот же объект, оставляя

только одну – с наивысшей уверенностью. Это необходимо для того, чтобы избежать дублирующих предсказаний одного и того же объекта.

Фреймворком для использования и обучения моделей YOLO является Ultralytics [72] – активно поддерживаемая библиотека, предоставляющая интерфейс для загрузки, настройки, обучения и инференса моделей семейства YOLO, включая последние версии, такие как YOLOv8. Последние версии этих моделей обучены на датасете COCO (Common Objects in Context) [61].

Для исследования путей решения поставленной задачи подходят следующие модели из семейства YOLO: YOLOv5s (small) – приоритет скорость (до 100+ FPS на GPU); YOLOv5m – баланс между скоростью и точностью; YOLOv8n (nano) – когда критична скорость и работа на слабых устройствах; YOLOv8s – универсальное решение с хорошей точностью. В данной работе дообучение будет проводиться на модели YOLOv8n (nano), так как она относится к последней версии семейства YOLO и обеспечивает наивысшую производительность среди лёгких моделей, что делает её особенно подходящей для задач реального времени.

2.1.2 Архитектура и особенности моделей RT-DETR

Модель RT-DETR была представлена в 2023 году как попытка объединить высокую точность трансформерных моделей для обнаружения объектов с возможностью работы в реальном времени [106]. RT-DETR относится к семейству моделей, построенных на архитектуре DETR (DEtection TRansformer), но в отличие от оригинального DETR, делает акцент на производительности и скорости, позволяя применять трансформеры в задачах детекции объектов в реальном времени. Модель относится к одноступенчатым детекторам, как и YOLO.

Основная идея RT-DETR заключается в отказе от традиционной якорной схемы (anchor-free) и использовании прямого сопоставления между выходами

сети и ground truth объектами с помощью алгоритма сопоставления Хангариана (Hungarian matching). Модель рассматривает задачу обнаружения как задачу прямой трансдукции: с входного изображения предсказывается фиксированное количество объектов без необходимости генерации кандидатов или применения non-maximum suppression (NMS).

Общая архитектура RT-DETR также делится на три ключевых компонента: Backbone, Encoder-Decoder Transformer, Prediction Head. В качестве backbone используется сверточная сеть (например, ResNet или CSPDarkNet), которая извлекает пространственные признаки. Затем они поступают в энкодер трансформера, который захватывает глобальный контекст, и в декодер, который использует learnable object queries для генерации финальных представлений объектов. Prediction head предсказывает координаты ограничивающих рамок, классы объектов и оценки уверенности на основе выходов трансформера.

Backbone извлекает признаки изображения и подаёт их в энкодер, где происходит агрегация глобального контекста. Декодер трансформера обрабатывает набор learnable queries (запросов объектов), которые взаимодействуют с признаками из энкодера, позволяя модели "найти" объекты на изображении. Prediction head представляет собой набор feed-forward сетей, которые преобразуют выходы декодера в bounding boxes и метки классов.

По сравнению с классическими CNN-базируемыми детекторами, такими как YOLO или SSD, RT-DETR обеспечивает более высокую точность, особенно в сложных сценах, за счёт внимательного контекстного анализа, характерного для трансформеров. При этом модель оптимизирована для реального времени и демонстрирует конкурентоспособную скорость (до 60 FPS на GPU) без потери качества детекции. С момента появления RT-DETR быстро получил широкое распространение благодаря своей универсальности и способности работать наравне с CNN-моделями в условиях ограниченных задержек.

Фреймворком для использования и обучения моделей RT-DETR является Hugging Face Transformers [73] – популярная и постоянно обновляемая

библиотека, обеспечивающая интерфейс для работы с трансформерными моделями, включая загрузку, настройку, дообучение и инференс моделей RT-DETR и других архитектур на базе трансформеров.

На текущий момент в библиотеке Hugging Face Transformers относительно немного моделей архитектуры RT-DETR, в то время как основные модели DETR и его вариации представлены достаточно широко. В данной работе использовалась самая популярная по скачиванию на Hugging Face среди RT-DETR моделей RT-DETR с backbone ResNet-101 variant vd, которая была разработана исследователями из Пекинского университета (Peking University) и предварительно обучена на датасетах COCO и Object365 [107].

В разделе 2.1 рассмотрены архитектурные особенности моделей YOLO и RT-DETR, а также их применимость к задаче распознавания объектов на дороге в реальном времени. Обе модели относятся к одноступенчатым решениям, ориентированным на высокую скорость инференса. YOLO – проверенный и широко применяемый подход, зарекомендовавший себя в многочисленных прикладных задачах. RT-DETR представляет собой более современную архитектуру на основе трансформеров, способную учитывать глобальный контекст сцены и обеспечивать конкурентную точность при сохранении высокой производительности. В подразделе проанализированы конкретные реализации моделей, доступные во фреймворках Ultralytics и Hugging Face Transformers. Для дообучения и дальнейшего сравнения в данной работе выбраны модели YOLOv8n и RT-DETR-R101-VD (Peking University), обладающие сбалансированными характеристиками точности и скорости.

2.2 Описание и предобработка данных дорожных сцен

В задачах распознавания объектов на видео с использованием нейронных сетей, таких как YOLO, RT-DETR, каждый кадр видео обрабатывается независимо. Эти модели не обладают встроенной памятью о предыдущих кадрах

и не используют временную информацию, что означает отсутствие межкадровой связности или трекинга объектов.

Поэтому в данной работе для дообучения и количественной оценки качества моделей используются размеченные изображения с аннотациями (bounding boxes и классы объектов). Для демонстрации работы моделей в условиях, приближенных к реальному времени, использовались видео без разметки, что допустимо, так как их цель – визуально проиллюстрировать способность моделей обнаруживать объекты в движущейся сцене).

Для проведения практической части использовался датасет BDD100K (Berkeley DeepDrive 100K) – один из крупнейших и наиболее разнообразных наборов данных, предназначенных для задач компьютерного зрения в области автономного вождения. [108]. На рисунке 9 приведено сравнение с другими аналогичными датасетами от авторов, а на рисунке 10 приведены представленные классы и количество объектов в них в оригинальном датасете:

	KITTI	Cityscapes	ApolloScape	Mapillary	BDD100K
# Sequences	22	~50	4	N/A	100,000
# Images	14,999	5000 (+2000)	143,906	25,000	120,000,000
Multiple Cities	No	Yes	No	Yes	Yes
Multiple Weathers	No	No	No	Yes	Yes
Multiple Times of Day	No	No	No	Yes	Yes
Multiple Scene types	Yes	No	No	Yes	Yes

Рисунок 9 – Сравнение BDD100K с некоторыми другими наборами данных об уличных сценах (приблизительный ориентир) [108]

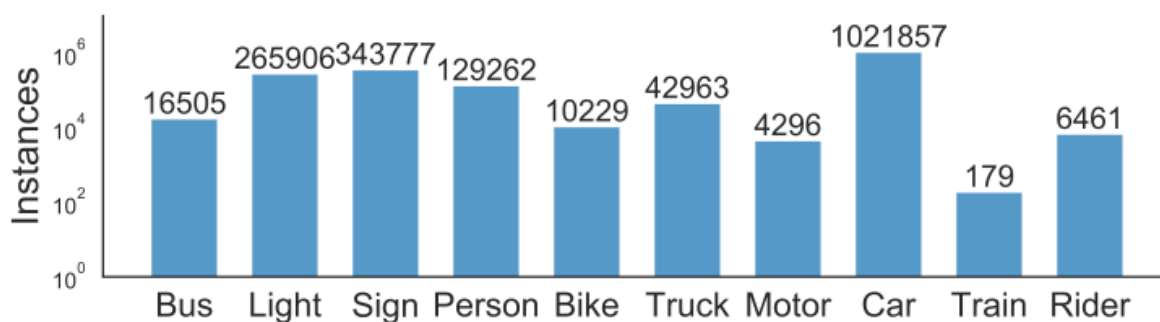


Рисунок 10 – Распределение объектов по различным типам объектов в датасете BDD100K [108]

Выбор BDD100K обусловлен следующими его преимуществами (Таблица 2). Датасет охватывает широкий спектр условий дорожной среды – от плотной городской застройки до пригородных и сельских участков, включая автомагистрали. Такое разнообразие обеспечит высокую обобщающую способность моделей, обученных на этих данных. Изображения собраны в различных городах и регионах США, что позволит моделям адаптироваться к различным архитектурным стилям, типам дорожной разметки, дорожным знакам и инфраструктурным особенностям. Кроме того, в датасет входят изображения, сделанные при разной погоде: ясной, облачной, дождливой, снежной и туманной, а также в разное время суток. Аннотации охватывают 10 классов, включая автомобили, пешеходов, велосипедистов, светофоры, дорожные знаки и другие типичные объекты дорожной обстановки, что соответствует основным требованиям к системам восприятия в задачах автономного управления. Изображения имеют разрешение 1280×720 пикселей, что обеспечивает оптимальный баланс между информативностью и скоростью обработки.

Помимо детекции объектов, в датасете представлены аннотации для сегментации, детекции дорожных событий и трекинга. Это делает BDD100K универсальным решением для мультизадачного обучения и дальнейшего расширения функциональности модели.

Таблица 2 – Преимущества датасета BDD100K

Преимущество	Описание
Разнообразие сцен	Городские улицы, пригороды, сельские дороги, автомагистрали
Географическая вариативность	Съемка в разных городах и регионах США
Погодные условия	Ясная, дождливая, снежная, туманная погода
Время суток	День, ночь, сумерки, рассвет
Актуальные классы объектов	10 классов: автомобили, пешеходы, светофоры, знаки и др.
Оптимальное разрешение	1280×720 – баланс между качеством и скоростью обработки
Дополнительные аннотации	Сегментация, трекинг, дорожные события

Таким образом, BDD100K предоставляет богатую и репрезентативную выборку дорожных сцен, позволяя обучать и оценивать модели в условиях, приближенных к реальной эксплуатации беспилотных автомобилей.

В работе использовался датасет BDD100K, размещённый на платформе Kaggle [109], что позволило упростить загрузку и подготовку данных. Эти изображения были использованы как для обучения, так и для количественной оценки точности и скорости распознавания объектов в условиях, приближённых к реальному времени.

Первом делом используемые данные были проверены на некорректные значения, дубли, пустые аннотации, изображения без аннотаций и очищены. Код по проверке и очистке данных приведен в приложении Б.

Проанализируем данные из BDD100K для обучения. Они содержат 70000 тренировочных изображений, 10000 – валидационных, с аннотациями в формате BDD100K (json), а также 20 000 неразмеченных тренировочных изображений. В таблице 3 представлены классы (за исключением drivable area и lane, поскольку они не относятся к задаче детекции объектов на дороге) и количества

соответствующих объектов. Наблюдается сильная несбалансированность классов:

Таблица 3 – Классы и количество объектов в них в тренировочном датасете BDD100K

Название класса BDD100K	Значение	Кол-во объектов
car	легковой автомобиль	713,211
traffic sign	дорожный знак	239,686
traffic light	светофор	186,117
person	пешеход	91,349
truck	грузовик	29,971
bus	автобус	11,672
bike	велосипед	7,210
rider	человек на двухколёсном транспорте (велосипеде или мотоцикле)	4,517
motor	мотоцикл	3,002
train	поезд	136

В связи с дисбалансом классов в тренировочных данных, было принято методологическое решение об исключении класса train (поезд), а также об объединении классов motor (мотоцикл) и bike (велосипед) в единый класс – двухколёсный транспорт.

Исключение класса train обосновано его крайне низкой представленностью в выборке – всего 136 объектов, что делает невозможным его полноценное обучение без риска переобучения и дисбаланса. Удаление данного класса не является критичным, так как поезда – нечастые объекты в типичной дорожной обстановке, особенно в контексте городского и загородного автономного вождения, где они практически не встречаются.

Решение об объединении классов `motor` и `bike` также основывается на ряде объективных соображений. Во-первых, мотоциклы и велосипеды имеют схожие визуальные характеристики и поведение на дороге. Во-вторых, объединение этих категорий в один класс способствует упрощению задачи классификации. Кроме того, данное объединение позволяет увеличить объём обучающих примеров для нового класса, что положительно сказывается на точности детекции.

Класс `rider` содержит относительно небольшое количество объектов (4 517), но играет вспомогательную роль при распознавании двухколёсного транспорта. Его исключение может негативно сказаться на качестве детекции классов `person` и `two_wheeler` в случае частых перекрытий. Код по преобразованию классов в тренировочных данных представлен в приложении Б.

Для смягчения дисбаланса классов были применены `oversampling` (оверсэмплинг) и `undersampling` (андэрсэмплинг) для малочисленных классов (`rider`, `two_wheeler`, `bus`) и многочисленных классов (`car`, `traffic sign`, `traffic light`) соответственно. Все изменения относительно структуры и объема классов в датасете отражены в таблице 4:

Таблица 4 – Комментарии по преобразованию классов из датасета BDD100K

Класс	Комментарий по использованию
<code>car</code>	Применен сильный андэрсэмплинг
<code>traffic sign</code> и <code>traffic light</code>	Применен несильный андэрсэмплинг вследствие андэрсэмплинга класса <code>car</code>
<code>bike</code> и <code>motor</code>	Объединены в общий класс <code>two_wheeler</code> , применен оверсэмплинг
<code>rider</code>	Оставлен, применен оверсэмплинг
<code>bus</code>	Применен оверсэмплинг
<code>train</code>	Удален
Остальные	Оставлены без изменений

Из-за того, что некоторые классы плохо представлены в датасете, модель будет игнорировать эти редкие классы или плохо их предсказывать. Первым шагом к лучшей балансировки данных стало увеличение представления редких классов: для каждого изображения, на котором есть хотя-бы один объект `rider`, `two_wheeler` или `bus`, была создана одна копия с новым именем формата: «{имя изображения}_copy.jpg», а также добавлена соответствующая аннотация для этой копии, аналогичная оригиналу. Важно понимать, что тем самым мы увеличиваем не только количество малочисленных классов, но и всех других, которые также попали на копируемые изображения. Код по применению оверсемплинга представлен в приложении В.

Переизбыток объектов класса `car` может привести к смещению обучения модели в его сторону (переобучению на доминирующем классе) и ухудшению распознавания менее представленных классов. В идеале было бы снизить количество объектов `car` примерно до 300-400 тыс. объектов – чтобы баланс был ближе к оптимальному. Однако важно не жертвовать мало представленными классами и сохранять реалистичность распределения. Поэтому была применена следующая стратегия андерсэмплинга в четыре этапа:

1. Удалены изображения, на которых присутствуют только объекты класса `car`;
2. Удалены изображения, на которых присутствуют только объекты класса `car` или `traffic sign` (но не только `traffic sign`);
3. Удалены изображения, на которых присутствуют только объекты класса `car` или `traffic light` (но не только `traffic light`);
4. Удалены изображения, на которых присутствуют только объекты класса `car` или `traffic light` или `traffic sign` (но не только `traffic light`, не только `traffic sign` и не только их комбинация).

При этом, на каждом этапе не удалялись изображения, на которых присутствовал хотя-бы один объект малочисленных классов (чтобы их не уменьшать еще больше). Кроме того, была введена проверка на то, чтобы

объектов класса car не осталось меньше 350000, а объектов traffic light и traffic sign – меньше 150000, чтобы не переборщить с их удалением (ни одна из этих проверок не сработала в процессе исполнения кода). Пример кода андэрсэмплинга для первого этапа представлен в приложении В. После всех изменений состав и численность объектов по классам в обучающем сете приведены в таблице 5:

Таблица 5 – Классы и количество объектов в них в тренировочном датасете BDD100K в рамках данной работы (после андэрсэмплинга и оверсэмплинга)

Класс BDD100K	Кол-во объектов до	Кол-во объектов после	Изменение	Комментарий
car	713 211	528 354	-184 857	Преобладает, но заметное снижение (~25% уменьшение). Остался крупнейшим классом, но переобучение на него теперь менее вероятно
traffic sign	239 686	195 987	-43 699	Хорошо представлен, чуть выше нормы, но допустимо
traffic light	186 117	179 105	-7 012	Хорошо представлен, всё ещё много, но уже в пределах разумного
person	91,349	130,512	+39 163	Сбалансирован. Класс важный, и его количество соответствует реальности.
truck	29,971	37,643	+7 672	Нормально представлен как изначально, так и теперь
bus	11,672	23,344	+11 672	Теперь сбалансирован
two_wheeler	10 212	20,424	+10 212	Теперь сбалансирован
rider	4 517	9,034	+4 517	Всё ещё относительно малочисленен

В результате проведенного андер- и оверсэмплинга доля класса car среди всех объектов снизилась примерно с 50% до 35%, а количество экземпляров малочисленных классов увеличилось в 2 раза. Дальнейшее уменьшение класса car могло бы повлечь за собой потерю важных, но малочисленных классов, что сделало бы выборку менее пригодной для обучения универсальной модели.

Применение взвешенных функций потерь либо стратифицированных батчей могло бы способствовать устранению оставшегося дисбаланса классов в анализируемых моделях. Однако в стандартных реализациях моделей от Ultralytics (YOLO) и Hugging Face (RT-DETR) отсутствуют подобные механизмы "из коробки", что ограничивает возможности прямой балансировки на этапе тренировки. Аналогично о применении focal loss: хотя в RT-DETR она используется по умолчанию, её модификация или адаптация под веса классов требует ручных изменений в коде.

Итоговый объем обучающих данных составил 52 736 изображений. Валидационных данных осталось столько же – 10 000.

После очистки данных и смягчения дисбаланса классов было необходимо привести их аннотации в форматы, необходимые для обучения моделей YOLO и RT-DETR соответственно.

Для YOLO используется простой текстовый формат аннотаций: для каждого изображения создается .txt файл, содержащий строки вида: <class_id> <x_center> <y_center> <width> <height>, где координаты нормированы от 0 до 1 относительно размеров изображения.

Для RT-DETR требуется COCO-формат: аннотации сохраняются в одном JSON-файле, включающем информацию об изображениях (images), объектах (annotations) и категориях (categories). Координаты объектов задаются как прямоугольники в формате [x, y, width, height] в пикселях.

Код по преобразованию аннотаций данных размещен в приложении Г.

Таким образом, в разделе 2.2 были описаны данные для обучения, проведена их очистка и анализ, в ходе которого был выявлен дисбаланс классов.

Для его смягчения были применены методы оверсэмплинга малочисленных классов и андерсэмплинга доминирующего. После чего данные были приведены к необходимому для обучения моделей формату.

2.3 Программные средства и библиотеки для реализации решения

Для обработки данных, загрузки предобученных нейросетевых моделей, их дообучения, оценки, тестирования и визуализации результатов для решения задачи распознавания объектов в реальном времени в рамках данного исследования использовались современные инструменты и программные средства, обеспечивающие удобную разработку, отладку и обучение нейросетевых моделей.

В качестве основного языка программирования выбран Python, что обусловлено его простотой, читаемостью синтаксиса и широким распространением в научном и инженерном сообществе. Python предоставляет обширный набор специализированных библиотек для задач компьютерного зрения, обработки изображений и построения нейросетевых архитектур [29].

Выбор языка программирования для создания нейронных сетей зависит: «... от конкретных потребностей проекта, а также предпочтений и навыков разработчика. Факторы, которые следует учитывать, включают простоту использования, производительность, гибкость и поддержку сообщества» [110].

Разработка и отладка кода осуществлялась в интерактивной среде Jupyter Notebook [111], которая позволяет быстро проверять гипотезы, визуализировать промежуточные результаты и удобно документировать процесс обучения.

Для загрузки и применения предобученных моделей семейства YOLO (You Only Look Once), в том числе современных версий YOLOv5 и YOLOv8, в данной работе использована библиотека ultralytics, разработанная одноимённой компанией Ultralytics [72]. Она предоставляет удобный высокоуровневый интерфейс для инициализации, инференса и обучения моделей, а также содержит встроенные механизмы обработки изображений и видеопотока.

Для загрузки и применения предобученных моделей объектного детектирования RT-DETR в данной работе использованы классы `RTDetrImageProcessor` и `RTDetrForObjectDetection` из библиотеки `transformers`, разработанной компанией Hugging Face [73]. Эти классы позволяют напрямую инициализировать соответствующий препроцессор и модель RT-DETR с предварительно обученными весами, опубликованными в Hugging Face Hub. Для процесса обучения фреймворк предоставляет класс `Trainer` – высокоуровневый интерфейс для организации тренировочного цикла.

Использование данных фреймворков позволило сосредоточиться на настройке параметров обучения и анализе результатов, минимизируя необходимость написания низкоуровневого кода.

Также в данной работе использовались вспомогательные библиотеки [104]:

1. Pillow – для загрузки и конвертации изображений;
2. OpenCV (cv2) – для отображения, предобработки изображений;
3. Matplotlib – для построения графиков и визуализации результатов;
4. torch – для построения и обучения нейронных сетей, работы с тензорами и оптимизации.

Обучение моделей производилось на видеокарте NVIDIA GeForce GTX 1630. Эта модель относится к начальному уровню дискретных GPU и оснащена 4 ГБ видеопамяти GDDR6, что ограничивает её возможности при работе с крупными нейронными сетями и большими размерами батча. GTX 1630 поддерживает технологию CUDA – платформа параллельных вычислений от компании NVIDIA, поддерживающая ускорение нейросетевых вычислений за счёт использования графических процессоров [104], однако не поддерживает автоматическое смешанное обучение (AMP, Automatic Mixed Precision), что лишает её преимуществ, связанных с экономией памяти и ускорением обучения при использовании FP16-вычислений.

Таким образом, выбранный стек инструментов (Python + фреймворки Ultralytics и Hugging Face + Jupyter + вспомогательные библиотеки) обеспечивает удобство, производительность и гибкость разработки. Он позволяет реализовать, обучать и тестировать модели распознавания объектов с высокой эффективностью и возможностью масштабирования под различные вычислительные ресурсы. Используемая для обучения видеокарта GTX 1630 может использоваться для обучения небольших моделей или для дообучения на ограниченных выборках, однако её производительность заметно уступает более мощным решениям и может ограничивать скорость и масштаб экспериментов.

Выводы по второй главе

В данной главе были обоснованы и подготовлены все необходимые компоненты для дообучения моделей распознавания дорожных объектов в реальном времени. Проведен обзор современных моделей детекции объектов – YOLO и RT-DETR, были выбраны их конкретные предобученные реализации из открытых библиотек. Особое внимание было уделено подготовке обучающих данных на основе датасета BDD100K. Для смягчения выявленного дисбаланса между классами применены методы оверсэмплинга и андерсэмплинга. Данные приведены к форматам, соответствующим требованиям моделей YOLO и RT-DETR. Также были рассмотрены программные инструменты и вычислительные ресурсы, использованные для реализации системы. Выбранный стек технологий обеспечивает удобность и высокую гибкость экспериментов, а используемая видеокарта NVIDIA GTX 1630 позволяет проводить дообучение легких моделей в условиях ограниченных ресурсов.

В целом, глава заложила теоретическую и практическую основу для последующего этапа – дообучения и сравнения нейросетевых моделей на специализированных дорожных данных.

3 Дообучение моделей и оценка их эффективности в задаче дорожной детекции в реальном времени

3.1 Дообучение моделей YOLOv8n и RT-DETR на датасете BDD100K

Для обучения модели YOLOv8 на датасете BDD100K, содержащем более 50 тысяч изображений, применяется полный fine-tuning без заморозки весов. Это значит, что все слои модели дообучаются на специфике дорожных сцен данного датасета, что позволяет максимально адаптировать предобученную модель к новой задаче. Большой объем данных обеспечивает устойчивое обновление весов и снижает риск переобучения, поэтому заморозка слоев не требуется. Настройки обучения представлены на рисунке 11:

```
from ultralytics import YOLO

# Загружаем предобученную модель yolov8n
model = YOLO("yolov8n.pt")

# Путь к файлу с конфигурацией данных (data.yaml), где указаны пути к train/val и классы
data_yaml_path = r"C:\Users\odara\Downloads\data\yolo_fine_tuning\data.yaml"

results = model.train(
    data=data_yaml_path,
    epochs=25,           # Кол-во эпох
    imgsz=768,           # Размер изображения
    batch=16,            # Для "средней" видеокарты
    name="yolov8n_bdd100k",
    project=r"C:\Users\odara\Downloads\data\yolo_fine_tuning",
    device=0,            # GPU
    weight_decay=0.0005, # L2 регуляризация
    patience=3,          # Раннее завершение при переобучении
    workers=8,           # Параллельная загрузка данных
    # freeze=10,          # Заморозка первых 10 слоев
    save_period=1,       # Сохранять чекпоинты каждую эпоху
    save=True
)
```

Рисунок 11 – Настройки дообучения YOLOv8n на датасете BDD100K

Для модели YOLOv8n было установлено 25 эпох обучения. Оптимизатор – SGD с параметрами $\text{momentum}=0.9$ и $\text{learning rate}=0.01$, подобранными автоматически системой Ultralytics YOLOv8 на основе анализа конфигурации модели, размера батча (16) и объёма данных (52 736). Такой выбор обусловлен тем, что SGD с импульсом часто обеспечивает более устойчивую сходимость и лучшую обобщающую способность на крупных датасетах при длительном обучении.

В процессе обучения YOLOv8n используется batch normalization, weight decay (L2 регуляризация) в оптимизаторе, а также стандартные аугментации данных. Применяется ранняя остановка (early stopping) для предотвращения переобучения. Скорость обучения регулируется встроенным lr scheduler. Функция потерь – стандартная для YOLO, комбинированная:

- loss по координатам боксов (обычно CIoU или GIoU loss);
- loss по объектности (objectness);
- loss по классам (кросс-энтропия).

Dropout обычно не применяется в сверточных сетях, особенно в современных архитектурах вроде YOLOv8, поскольку batch normalization уже эффективно борется с переобучением. L1 регуляризация редко используется в задачах детекции объектов, так как она может слишком сильно обнулять веса, ухудшая качество обучения. Поэтому при дообучении YOLOv8 эти методы в данной работе не задействованы.

Размер изображения 768×768 пикселей выбран как компромисс между качеством распознавания и скоростью обработки. Он достаточно велик для сохранения важных визуальных деталей объектов на дороге, что способствует более точному распознаванию, особенно мелких или удалённых объектов. В то же время, этот размер не приводит к чрезмерной нагрузке на видеопамять и незначительно влияет на скорость инференса по сравнению с меньшими размерами. При этом стоит отметить, что стандартный размер изображений для предобученных моделей YOLO составляет 640×640 пикселей, однако

использование увеличенного разрешения может дать прирост качества при незначительном увеличении вычислительных затрат.

Выбрано значение workers=8, поскольку у используемой системы 14 ядер и 20 потоков. Это примерно половина от общего количества потоков, что позволяет эффективно использовать многозадачность без чрезмерной нагрузки на процессор. Такой баланс обеспечивает оптимальную загрузку GPU и диска при параллельной обработке данных, снижая вероятность узких мест и простоев во время обучения модели.

Визуализация процесса обучения YOLOv8n встроена автоматически (Рисунок 12 и 13):

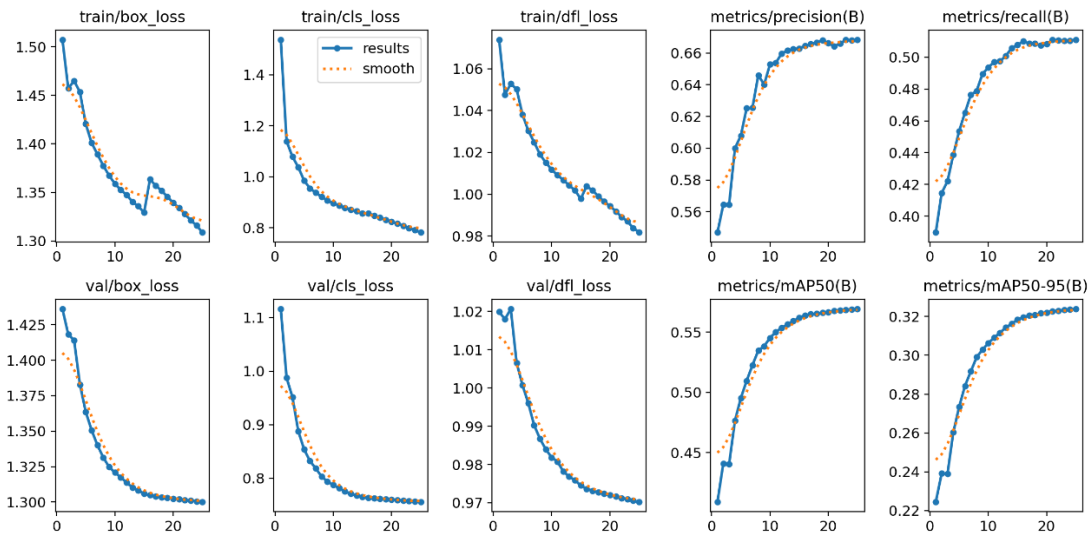


Рисунок 12 – Процесс обучения YOLOv8n

23/25	5.36G	1.321	0.7992	0.987	270	768: 100%	3296/3296	[2:34:52<00:0
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100%	313/313 [05:
	all	10000	185511	0.668	0.51	0.569	0.323	
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size		
24/25	5.63G	1.316	0.7911	0.9838	326	768: 100%	3296/3296	[2:26:17<00:0
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100%	313/313 [05:
	all	10000	185511	0.668	0.511	0.569	0.324	
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size		
25/25	5.38G	1.309	0.7818	0.9818	330	768: 100%	3296/3296	[2:40:32<00:0
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100%	313/313 [05:
	all	10000	185511	0.668	0.511	0.569	0.324	

Рисунок 13 – Конечные эпохи обучения YOLOv8n

Для обучения модели RT-DETR-R101-VD на датасете BDD100K также был использован полный fine-tuning. Настройка дообучения для модели RT-DETR-R101-VD представлена на рисунке 14.

```
from transformers import TrainingArguments

training_args = TrainingArguments(
    output_dir="C:/Users/odara/Downloads/data/rtdetr_fine_tuning",
    num_train_epochs=10,
    per_device_train_batch_size=2,
    per_device_eval_batch_size=2,
    learning_rate=2e-5,
    weight_decay=0.01, # Регуляризация

    dataloader_num_workers=0,

    save_strategy="epoch",
    save_total_limit=1,
    logging_dir="C:/Users/odara/Downloads/data/rtdetr_fine_tuning/logs",
    logging_strategy="steps",
    logging_steps=100,
    disable_tqdm=False,

    eval_strategy="epoch",
    load_best_model_at_end=True,
    metric_for_best_model="loss",
    greater_is_better=False,

    # fp16=True,

    # Дополнительные параметры для стабильности и производительности
    gradient_accumulation_steps=1,
    dataloader_pin_memory=False
)
```

Рисунок 14 – Настройки дообучения модели RT-DETR-R101-VD на датасете BDD100K

Для загрузки и процессора изображений были использованы классы RTDetrForObjectDetection и RTDetrImageProcessor (Рисунок 15). Для обучения – класс Trainer (Рисунок 16). Код создания кастомных датасетов для загрузки в Trainer представлен в приложении Д.

```

from transformers import RTDetrImageProcessor, RTDetrForObjectDetection

processor = RTDetrImageProcessor.from_pretrained(
    "PekingU/rtdetr_r101vd_coco_o365",
    size={"height": 512, "width": 512}
)
model = RTDetrForObjectDetection.from_pretrained("PekingU/rtdetr_r101vd_coco_o365")

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

```

Рисунок 15 – Загрузка модели RT-DETR-R101-VD и процессора изображений

Для обучения RT-DETR-R101-VD используется оптимизатор AdamW, хорошо подходящий для трансформерных архитектур благодаря отдельной обработке весов и смещений. В качестве функции потерь применяется совокупная loss-функция, включающая:

- hungarian loss для сопоставления предсказаний и истинных объектов (основа DETR-подхода),
- box loss (L1 + GIoU) для координат прямоугольников,
- classification loss (CrossEntropy) для распознавания классов объектов.

Такая композиция позволяет модели точно локализовывать и классифицировать объекты без применения NMS.

Изображения приводились к разрешению 512×512 пикселей – это компромисс между сохранением детализации объектов и производительностью при использовании ограниченного GPU. Данный размер позволяет модели эффективно воспринимать как крупные, так и мелкие дорожные объекты при разумном времени обработки одного кадра.

В качестве параметров обучения были заданы:

- количество эпох: 10 – достаточное число для дообучения предобученной модели на специфическом наборе дорожных изображений;
- размер пакета (batch size): 2 – обусловлен ограниченным объемом видеопамяти доступной GPU, а также особенностями реализации модели;

- шаг обучения (learning rate): 2×10^{-5} – небольшой шаг, позволяющий избежать разрушения ранее обученных признаков;
- снижение переобучения: достигалось за счет применения L2-регуляризации (weight decay = 0.01).

Трансформеры, несмотря на более высокую вычислительную сложность, быстрее усваивают семантические зависимости и требуют меньше эпох при дообучении на COCO-подобных данных. Количество эпох для RT-DETR было выбрано так, чтобы общее время обучения сопоставлялось с обучением YOLOv8n на 25 эпохах.

Особенности аппаратного обеспечения также учитывались в настройках. Поскольку обучение выполнялось с использованием непроизводительной GPU, были задействованы меры по снижению нагрузки: умеренное разрешение изображений, минимальный batch size, отключение накопления градиентов, и сохранение ограниченного числа контрольных точек. Полупроизвольная точность (FP16) была не активирована, поскольку стабильная работа в таком режиме требует более мощной графической карты.

В параметрах загрузки данных значение num_workers было установлено равным 0, так как при использовании RT-DETR-R101-VD в среде Windows возникали проблемы с запуском обучения при параллельной загрузке. Это решение обеспечило стабильность процесса без значительной потери в производительности. Также параметр pin_memory был отключён, поскольку при его включении возникала ошибка: в данной конфигурации загрузки данных он ожидался только для использования с CPU. Для обеспечения корректной работы загрузчика и избежания сбоев параметр был установлен в значение False.

Таким образом, конфигурация обучения была сбалансирована с учетом архитектурных особенностей модели, ограничений оборудования и требований к точности. Подход обеспечил приемлемое время обучения при сохранении высокого качества обнаружения объектов, что критично для применения модели

в условиях реального времени на борту автономных транспортных средств. Процесс обучения представлен на рисунках 16 и 17.

Step	Training Loss	Validation Loss
10000	13.218400	11.252387
20000	12.964100	10.821891
30000	12.477300	10.735963
40000	12.475600	10.736838
50000	12.381400	10.687516
60000	12.024200	10.510774
70000	12.013600	10.707314

Рисунок 16 – Начальные шаги обучения RT-DETR-R101-VD

Epoch	Training Loss	Validation Loss
8	10.760900	10.198097
9	10.849000	10.234483
10	10.669700	10.179273

Рисунок 17 – Последние эпохи обучения RT-DETR-R101-VD

В данном подразделе были дообучены модели YOLO8n и RT-DETR-R101-VD. В обоих случаях дообучение было достаточно стабильным и сопровождалось снижением функции потерь. Обучение производилось в условиях ограниченных вычислительных возможностей, поэтому стоит сделать оговорку, что для достижения более высокой потенциальной точности моделей стоило бы увеличить количество эпох и размер обучающих изображений при более высоких вычислительных ресурсах. Модель YOLO обучались около 72-х часов, RT-DETR – около 90 часов.

3.2 Сравнительный анализ эффективности дообученных моделей

Для оценки модели и расчёта метрик точности (precision, recall, mAP) используются валидационные данные, так как тестовые данные BDD100K не содержат аннотаций.

Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100%
all	10000	185511	0.669	0.511	0.569	0.324
car	9879	102506	0.79	0.672	0.756	0.467
traffic sign	8221	34908	0.707	0.515	0.589	0.306
traffic light	5653	26885	0.695	0.5	0.56	0.21
person	3220	13262	0.727	0.491	0.582	0.288
truck	2689	4245	0.632	0.532	0.586	0.422
bus	1242	1597	0.569	0.548	0.586	0.451
two_wheeler	856	1459	0.601	0.433	0.465	0.228
rider	515	649	0.627	0.398	0.428	0.218

Рисунок 18 – Метрики точности дообученной модели YOLOv8n по классам

Speed: 0.4ms preprocess, 8.5ms inference, 0.0ms loss, 0.7ms postprocess per image

Рисунок 19 – Скорость инференса дообученной модели YOLOv8n

В результате обучения модели YOLOv8n на 25 эпохах были получены итоговые метрики по задаче распознавания объектов на дороге на валидационной выборке из 10 000 изображений (185 511 объектов) (Рисунок 18). Основная цель заключалась в оценке применимости лёгкой модели (YOLOv8n) для задач реального времени в условиях ограниченных вычислительных ресурсов, в частности – для управления беспилотным транспортом.

Общие метрики по всем классам составили: точность (precision) – 0.669, полнота (recall) – 0.511, mAP@0.5 – 0.569 (Рисунок 20), и mAP@0.5:0.95 – 0.324. Это демонстрирует приемлемый уровень качества, особенно с учётом компактности и скорости модели. Среди отдельных классов лучшие результаты достигнуты по ключевым объектам дорожной сцены: автомобили (car) –

$mAP@0.5 = 0.756$, дорожные знаки (traffic sign) – 0.589 и пешеходы (person) – 0.582. Эти объекты наиболее критичны с точки зрения безопасности движения.

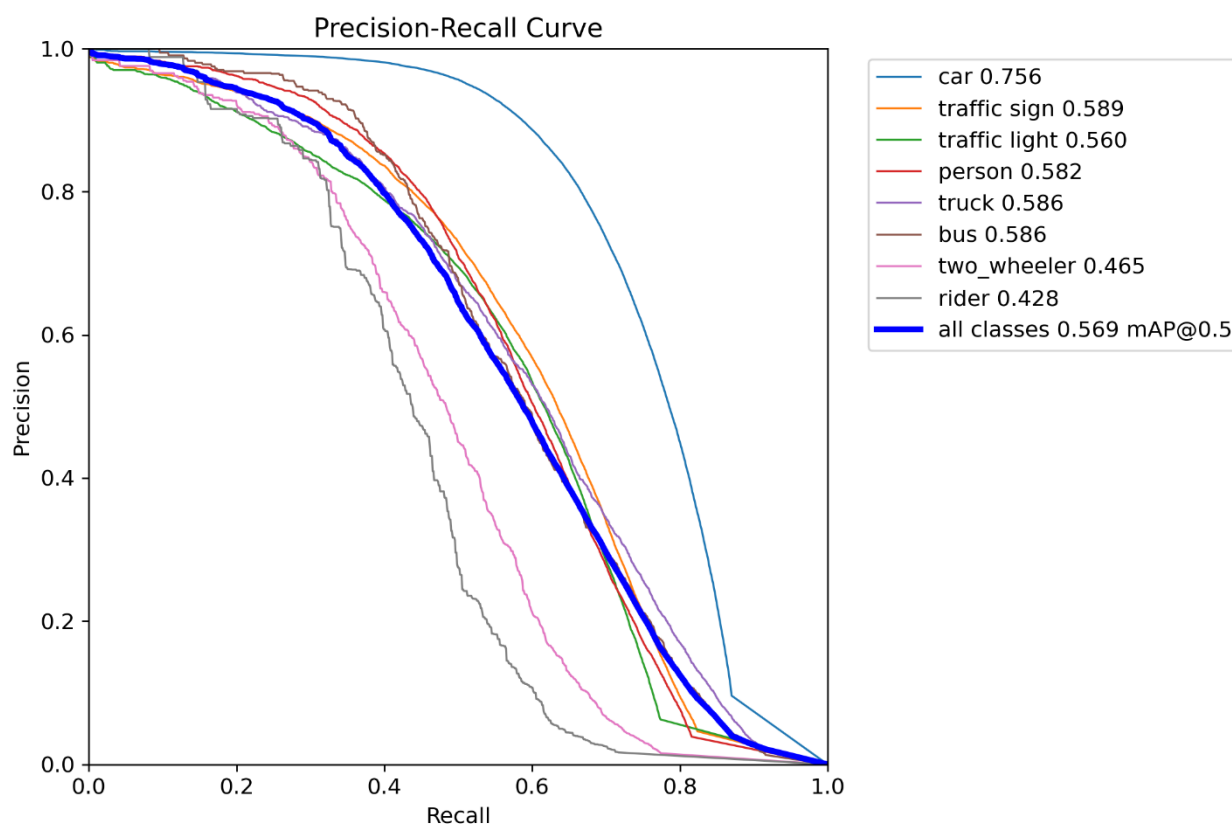


Рисунок 20 –PR-curve для YOLOv8n по классам

Более низкие показатели наблюдаются у классов traffic light (0.560 $mAP@0.5$), truck (0.586), bus (0.586) и особенно two_wheeler (0.465) и rider (0.428). Наихудшие значения – у класса «rider» с $recall = 0.398$ и $mAP@0.5:0.95 = 0.218$, что может быть связано с ограниченным количеством примеров, малым размером объектов на изображениях и визуальной неоднородностью. Однако в ряде приложений, особенно при использовании более лёгких моделей в системах реального времени, такие классы могут считаться второстепенными.

Средние значения точности (Precision) и полноты (Recall) по всем классам составили 0.669 и 0.511 соответственно. Это говорит о том, что модель демонстрирует приемлемую точность распознавания объектов, однако всё ещё

пропускает значительное количество целевых объектов, особенно при наличии сложных условий съёмки, мелких размеров объектов или перекрытий.

Особенно заметно снижение полноты у классов `two_wheeler` (0.433) и `person` (0.491), что может быть вызвано недостаточным числом обучающих примеров, визуальной сложностью этих объектов, а также их частичным перекрытием другими элементами сцены.

Согласно журналу обучения, среднее время обработки одного изображения моделью YOLOv8n составляло (Рисунок 19):

- 0.4 мс на этап предобработки (preprocess);
- 8.5 мс – на непосредственный инференс (inference);
- 0.7 мс – на постобработку выходных данных (postprocess).

Суммарное время обработки одного изображения составляет приблизительно 9.6 мс, что соответствует производительности более 100 кадров в секунду (FPS) при inference на GPU. Это подтверждает высокую скорость работы модели и её пригодность для использования в системах реального времени, включая задачи управления беспилотным автомобилем.

Финальный вес модели YOLOv8n после дообучения на датасете BDD100K составил 5.94 МБ, что лишь незначительно превышает размер предобученной версии (~4.7 МБ). Это подчёркивает компактность архитектуры и её пригодность для развёртывания на устройствах с ограниченными ресурсами, включая системы реального времени на борту беспилотных автомобилей.

Оценка результатов дообученной модели RT-DETR-R101-VD на валидационной выборке датасета BDD100K показала, что модель достигает значения $mAP@0.50 = 0.537$, что можно считать удовлетворительным уровнем точности для применения в задачах реального времени (Рисунок 21). Особенно высокая точность наблюдается для таких классов, как «car» (0.7343), «truck» (0.5750) и «bus» (0.5712), что критически важно для обеспечения надёжного обнаружения основных транспортных средств в дорожной среде. Менее точные результаты были получены по классам «traffic light» (0.4909), «rider» (0.4220) и

«two_wheeler» (0.4467), что соответствует общим трудностям детекции мелких и частично перекрытых объектов (Рисунок 22).

Обобщённая метрика $mAP@[0.5:0.95]$ составила 0.307, что указывает на более скромную способность модели к точному локализованному распознаванию на высоких порогах IoU (Рисунок 23).

Помимо метрик точности, модель продемонстрировала и приемлемый уровень полноты (Recall). Average Recall (AR)@[IoU=0.50:0.95] при maxDets=100 составил 0.454, что говорит о способности модели находить почти половину всех объектов на изображениях. Особенно высокое значение Recall зафиксировано для крупных объектов (AR = 0.758), что важно для обнаружения основных участников дорожного движения, тогда как Recall для мелких объектов остаётся на более низком уровне (AR = 0.252).

Average Precision	(AP) @[IoU=0.50:0.95 area= all maxDets=100]	= 0.307
Average Precision	(AP) @[IoU=0.50 area= all maxDets=100]	= 0.537
Average Precision	(AP) @[IoU=0.75 area= all maxDets=100]	= 0.291
Average Precision	(AP) @[IoU=0.50:0.95 area= small maxDets=100]	= 0.096
Average Precision	(AP) @[IoU=0.50:0.95 area=medium maxDets=100]	= 0.365
Average Precision	(AP) @[IoU=0.50:0.95 area= large maxDets=100]	= 0.634
Average Recall	(AR) @[IoU=0.50:0.95 area= all maxDets= 1]	= 0.217
Average Recall	(AR) @[IoU=0.50:0.95 area= all maxDets= 10]	= 0.410
Average Recall	(AR) @[IoU=0.50:0.95 area= all maxDets=100]	= 0.454
Average Recall	(AR) @[IoU=0.50:0.95 area= small maxDets=100]	= 0.252
Average Recall	(AR) @[IoU=0.50:0.95 area=medium maxDets=100]	= 0.542
Average Recall	(AR) @[IoU=0.50:0.95 area= large maxDets=100]	= 0.758

Рисунок 21 – Метрики AP и AR модели RT-DETR-R101-VD после дообучения

```
car           : AP@0.50 = 0.7343
traffic sign  : AP@0.50 = 0.5456
traffic light : AP@0.50 = 0.4909
person        : AP@0.50 = 0.5133
truck         : AP@0.50 = 0.5750
bus           : AP@0.50 = 0.5712
two_wheeler   : AP@0.50 = 0.4467
rider         : AP@0.50 = 0.4220
```

без класса rider метрика немного выше – 0.5539, вместо 0.5374

Рисунок 22 – AP@0.50 модели RT-DETR-R101-VD после дообучения

car	: AP@[IoU=0.50:0.95] = 0.4386
traffic sign	: AP@[IoU=0.50:0.95] = 0.2796
traffic light	: AP@[IoU=0.50:0.95] = 0.1747
person	: AP@[IoU=0.50:0.95] = 0.2609
truck	: AP@[IoU=0.50:0.95] = 0.4174
bus	: AP@[IoU=0.50:0.95] = 0.4409
two_wheeler	: AP@[IoU=0.50:0.95] = 0.2201
rider	: AP@[IoU=0.50:0.95] = 0.2203

без учёта класса rider метрика немного выше: 0.3189, вместо 0.3191 с ним

Рисунок 23 – AP@[0.5:0.95] модели RT-DETR-R101-VD после дообучения

Дообученная модель RT-DETR-R101-VD дообученная на датасете BDD100K, имеет итоговый вес 293,1 МБ. В процессе оценки её производительности на изображениях из валидационного множества (с приведением разрешения к 512×512 на этапе препроцессинга) было получено среднее полное время обработки одного изображения, включая препроцессинг, инференс и постпроцессинг. Оно составило 158.67 мс, что эквивалентно примерно 6.3 кадра в секунду (FPS). Текущая скорость недостаточна для плавного управления в реальном времени, особенно в критичных задачах беспилотного вождения, что говорит о необходимости рассмотрения более лёгких архитектур RT-DETR с меньшим бэкбоном в дальнейших исследованиях.

Для более объективной оценки качества полученных моделей дообученных на BDD100K была проведена сравнительная проверка с типичными значениями точности, достигаемыми в исследовательских работах. В таблице 6 приведены ориентировочные значения mAP по метрике mAP@0.5, поскольку она лучше отражает прикладную точность моделей в задачах реального времени, где важна своевременная и уверенная локализация объектов, а не строгая привязка к точному перекрытию. Метрика mAP@[0.5:0.95] же является более строгой усреднённой метрикой и традиционно используется в научных бенчмарках (например, COCO).

Следует заметить, что технические характеристики моделей или систем, используемых в реальных прикладных решениях беспилотного управления

транспортом, как правило, редко публикуются в открытом доступе, так как сведения о структуре моделей, точности распознавания, объеме весов и конкретных параметрах обучения являются частью закрытых корпоративных технологий.

Таблица 6 – Сравнение дообученных моделей YOLOv8n и RT-DETR-R101-VD с аналогичными исследовательскими результатами

Модель	Размер (вес)	mAP@0.5	Датасет обучения
yolov8n (в данной работе)	~5.94 МБ	0.569	BDD100K (дообучение), 768×768
rt detr_r101vd_coco_o365 (в данной работе)	~293.1 МБ	0.537	BDD100K (дообучение), 512×512
yolov8n ориг. [72]	~4.7 МБ	0.672	COCO 2017, 640×640
rt detr_r101vd_coco_o365 ориг. [107]	~ 307 МБ	0.746	COCO 2017 + Objects365, 640×640
yolov8n [112]	~3.8 МБ	0.201	Подмножество BDD100K (дообучение), 640×640
yolov5s [113]	~14.3 МБ	0.558	BDD100K (дообучение), 640×640

Сравнение оригинальных моделей YOLOv8n и RT-DETR-R101-VD, обученных на универсальных датасетах COCO и Objects365, показывает, что они достигают высоких показателей mAP@0.5 (соответственно 0.672 и 0.734) при разрешении входных изображений 640×640. Размеры моделей изначально существенно различаются: YOLOv8n весит около 5 МБ, в то время как RT-DETR-R101-VD значительно крупнее – порядка 76 МБ.

При дообучении на специализированном датасете BDD100K обе модели показали снижение точности по сравнению с оригинальными версиями. Это закономерно, поскольку специализированный дорожный датасет отличается по

составу, сложности и размеру изображений, а также предъявляет более узкие требования к детекции именно дорожных объектов.

Несмотря на снижение mAP по сравнению с оригинальными моделями, точность, достигнутая в данной работе, остаётся конкурентоспособной и адекватной в сравнении с аналогичными работами по дообучению моделей YOLO на датасете BDD100K (работ по дообучению RT-DETR на BDD100K найдено не было). Для YOLOv8n показатель mAP выше, чем в других работах с подмножествами BDD100K, что подтверждает эффективность выбранного подхода. Точность RT-DETR-R101-VD ($mAP@0.5 = 0.537$) немного ниже, чем у приведенной дообученной YOLOv5s ($mAP@0.5 = 0.558$), что вероятно связано с более низким разрешением обучающих изображений.

Можно также отметить некоторые работы, в которых модель YOLOv8n применялась к задачам, схожим с рассматриваемой в данной работе: так, YOLOv8n с весом около 3.2 МБ достигла значения $mAP@0.5 \approx 0.49$ на пользовательском датасете для обнаружения дорожных ям с разрешением изображений 1280×1280 [114], а также продемонстрировала $mAP@0.5 \approx 97.4$ на датасете JUIVCDv1, предназначенном для детекции транспортных средств [115]. Что касается модели RT-DETR-R101-VD, на момент написания настоящей работы не было обнаружено публикаций, в которых бы приводились результаты ее дообучения (а также каких-либо других моделей семейства RT-DETR) на специализированных дорожных датасетах, сопоставимых с условиями данной задачи.

В рамках проведённого эксперимента были сравнены два подхода к детекции объектов на дороге в реальном времени: лёгкая сверточная модель YOLOv8n и трансформерная модель RT-DETR с бэкбоном ResNet-101-VD, обе дообученные на подвыборке датасета BDD100K. По итогам тестирования точности на метрике $mAP@0.5$, YOLOv8n достигла значения 0.598, в то время как RT-DETR-R101-VD – 0.554. При этом YOLOv8n обучалась на изображениях с разрешением 768×768 в течение 25 эпох (общей продолжительностью около 72

часов), а RT-DETR-R101-VD – на изображениях 512×512 и только 10 эпох (около 90 часов), что обусловлено значительно большей вычислительной сложностью трансформерной архитектуры.

Полученные результаты демонстрируют, что при ограниченных вычислительных ресурсах и временных ограничениях лёгкие модели, такие как YOLOv8n, могут достигать более высоких показателей точности за счёт возможности увеличения количества эпох и использования изображений большего разрешения. Несмотря на это, RT-DETR-R101-VD обладает потенциально более высокой точностью при соответствующих условиях обучения, таких как увеличение числа эпох, размеров входных изображений или использование более мощных вычислительных платформ.

С точки зрения практического применения, YOLOv8n имеет существенное преимущество по ресурсной эффективности. Вес модели составляет всего 5.94 МБ, тогда как у RT-DETR-R101-VD он достигает 293.1 МБ, что критично для развёртывания на edge-устройствах. Кроме того, среднее время инференса YOLOv8n на изображениях 768×768 составило 9.6 мс, что позволяет достигать более 100 FPS и обеспечивает стабильную работу в режиме реального времени. В то же время RT-DETR на изображениях 512×512 показала среднее время полного инференса 158.7 мс, что соответствует лишь 6.3 кадра в секунду (FPS) – недостаточно для задач обработки видеопотока в реальном времени. Таким образом, использование RT-DETR-R101-VD в рамках заявленной цели возможно лишь при наличии мощного GPU или адаптации архитектуры (например, через применение более лёгких вариантов модели).

В целом, обе модели достигли приемлемого уровня точности ($mAP@0.5 > 0.5$) на открытом датасете, подтверждая свою применимость в задачах детекции дорожных объектов. Модель YOLOv8n продемонстрировала высокую практическую пригодность для использования в системах реального времени: она обеспечивает уверенное распознавание ключевых объектов дорожной сцены при умеренных требованиях к ресурсам. RT-DETR-R101-VD также показала себя

как перспективное решение для задач, где приоритет отдается качеству детекции, особенно для крупных или важных объектов, однако в текущей реализации её применение в реальном времени ограничено.

3.3 Демонстрация моделей и рекомендации для дальнейших исследований

В качестве примеров практической демонстрации работы дообученных моделей YOLOv8 и RT-DETR-R101-VD в задаче распознавания дорожных объектов используются изображения из валидационного набора BDD100K, охватывающие различные условия окружающей среды – дневное и ночное время, а также сцены с повышенной сложностью восприятия, включая нестандартные дорожные ситуации. Такой подход позволит не только визуально оценить качество распознавания, но и выявить сильные и слабые стороны каждой из моделей в типичных сценариях, с которыми может столкнуться беспилотный автомобиль в реальных условиях.



Рисунок 25 – Сопоставление выходов модели RT-DETR-R101-VD с реальной аннотацией объектов

```

Evaluation for image: b2a8e8b4-50058f09.jpg
Total GT objects      : 45
Predicted objects     : 30
Matched objects (IoU ≥ 0.5) : 21
Precision              : 0.70
Recall                 : 0.47
Correct class ratio    : 0.95
  
```

Рисунок 26 – Анализа точности предсказаний модели RT-DETR-R101-VD для конкретного изображения

На основе проведённого анализа результатов распознавания, а также выявленных ошибок в сложных и нестандартных дорожных ситуациях, были сформулированы следующие рекомендации, направленные на повышение точности и надёжности работы нейросетевых моделей в реальных условиях:

1. Доразметка "трудных" примеров. Дополнение обучающего набора вручную отобранными изображениями, на которых модели демонстрируют слабые результаты (частичное перекрытие объектов, ночные сцены, засветка фар и пр.), может значительно повысить устойчивость моделей к подобным случаям.
2. Использование продвинутых методов аугментации данных. Внедрение техник аугментации, имитирующих погодные эффекты (дождь,

туман, снег), блики, искажения и шум, позволит увеличить обобщающую способность моделей и подготовить их к реальным условиям эксплуатации.

3. Целенаправленное дообучение на сложных условиях. После сбора трудных примеров возможно проведение этапа таргетированного дообучения (fine-tuning) только на этих подмножествах.

4. При наличии более производительного оборудования представляется целесообразным провести углублённую оптимизацию моделей. Это может включать увеличение разрешения входных изображений, обучение в течение большего числа эпох (при контроле переобучения), а также использование более сложных и ёмких архитектур, таких как YOLOv8m/h. Указанные меры позволяют повысить точность распознавания, особенно в сложных сценах с высокой плотностью объектов и мелкими деталями.

5. Для повышения полноты распознавания целесообразно дополнить обучающий датасет примерами редких, но критически важных объектов, таких как спецтехника, мотоциклисты, эвакуаторы, прицепы и нестандартные дорожные знаки. Такие объекты могут встречаться нечасто, но их корректное распознавание необходимо для безопасного функционирования системы в нестандартных дорожных ситуациях.

6. Рассмотрение дополнительных классов объектов, часто встречающихся на дороге. В ходе анализа ошибок было установлено, что модель может ошибочно классифицировать вспомогательные элементы дорожной инфраструктуры, например, направляющие столбики или бордюры, как дорожные знаки или другие объекты. Включение таких элементов в обучающий набор в качестве отдельных, явно различаемых классов поможет существенно сократить количество ложных срабатываний и повысить точность классификации в реальных дорожных условиях.

7. Создание локализованного датасета для российских условий. Адаптация моделей под реалии российских дорог, включая особенности инфраструктуры, типов транспорта и дорожных знаков, может быть достигнута

путём формирования специализированного датасета, отражающего отечественные условия движения.

8. Исследование и внедрение специализированных моделей для сложных условий. В наиболее трудных сценариях (ночь, осадки, засветка, плохая видимость) может потребоваться использование более тяжёлых моделей, способных учитывать временную информацию (например, видеопоследовательности) или интегрировать дополнительные сенсорные данные. Также представляют интерес архитектуры, ориентированные на устойчивость к шумам, бликованию и другим искажениям.

В данном разделе была проведена демонстрация работы дообученных моделей YOLOv8 и RT-DETR-R101-VD на конкретных изображениях, охватывающих различные условия дорожной обстановки. Выполнена визуальная оценка качества распознавания, что позволило проанализировать поведение моделей в типичных и сложных сценариях. На основе полученных результатов были сформулированы рекомендации по возможным направлениям повышения точности и устойчивости моделей в будущих исследованиях.

Выводы по третьей главе

В третьей главе представлено экспериментальное исследование эффективности выбранных нейросетевых моделей в задаче распознавания объектов на дороге в режиме реального времени для управления беспилотным автомобилем. В начале главы подробно описан процесс дообучения моделей YOLOv8n и RT-DETR-R101-VD на адаптированном датасете BDD100K.

Далее проведена оценка и сравнение моделей по метрикам точности и производительности между собой, а также с другими современными решениями и бенчмарками, что позволило оценить их пригодность для использования в системах распознавания дорожных объектов в реальном времени.

Была проведена демонстрация работы дообученных моделей YOLOv8n и RT-DETR на конкретных изображениях, охватывающих различные условия дорожной обстановки, для визуальной и количественной оценки качества распознавания. На основе визуального анализа результатов распознавания были выдвинуты предложения и рекомендации по возможным путям повышения точности.

ЗАКЛЮЧЕНИЕ

Задача распознавания объектов на дороге является одной из ключевых в контексте обеспечения безопасного и автономного управления транспортным средством. Её успешное решение позволяет системе вождения в реальном времени обнаруживать и классифицировать важные элементы дорожной сцены, такие как другие автомобили, пешеходы, велосипеды, дорожные знаки, светофоры и т.д. Это, в свою очередь, даёт возможность принимать обоснованные решения по управлению транспортом: торможение, перестроение, снижение скорости или экстренное реагирование.

Целью настоящей магистерской диссертации было исследование и практическое применение моделей глубокого обучения для разработки решений, способных распознавать дорожные объекты в видеопотоке в реальном времени.

Задачи, выполненные для достижения цели:

1. Изучены теоретические основы нейронных сетей и исследования применению глубокого обучения к распознаванию объектов на дороге;
2. Проанализированы современные архитектуры глубоких нейронных сетей и инструменты для их обучения и оценки в задачах распознавания объектов в реальном времени.
3. Разработана методология дообучения выбранных моделей на специализированном датасете дорожных сцен и выполнена их адаптация;
4. Проведены тестирование и сравнительный анализ точности и производительности дообученных моделей.

В первой главе работы были рассмотрены теоретические основы функционирования нейронных сетей, их структура, методы обучения, а также основные метрики, применяемые для оценки качества распознавания объектов – метрики точности и производительности моделей. Проведён литературный обзор научных исследований, в которых применялись нейросетевые подходы к задаче распознавания объектов на дороге. На основе обзора был сделан вывод и

том, что модели семейства YOLO широко применяются в задачах дорожной детекции, в то время как трансформерные модели, такие как RT-DETR, только начали находить свое применение в задачах систем распознавания для управления беспилотным автомобилем.

Во второй главе были подробно рассмотрены архитектуры YOLO и RT-DETR и выбраны конкретные их реализации для работы в реальном времени для дообучения. Для адаптации моделей к дорожной специфике были выбраны данные BDD100K, которые были преобразованы для дообучения и в конечном итоге содержали восемь ключевых классов дорожных объектов: car, traffic sign, traffic light, person, truck, bus, two_wheeler и rider. Кроме того, были выбраны и описаны инструменты разработки: язык программирования – Python, среда разработки – Jupyter Notebook, фреймворки для загрузки и обучения моделей – Ultralytics и Hugging Face.

В рамках третьей главы были сравнены два подхода к детекции объектов на дороге в реальном времени: компактная сверточная модель YOLOv8n и трансформерная модель RT-DETR с бэкбоном ResNet-101-VD. Обе модели дообучались на датасете BDD100K. По метрике mAP@0.5 YOLOv8n достигла 0.598, а RT-DETR – 0.554. При этом YOLOv8n обучалась 25 эпох на изображениях 768×768 (72 часа), в то время как RT-DETR – 10 эпох на 512×512 (90 часов), что отражает высокую вычислительную нагрузку трансформеров.

Результаты показали, что при ограниченных ресурсах лёгкие модели, такие как YOLOv8n, могут быть более эффективными за счёт возможности увеличения количества эпох и размера изображений. Тем не менее RT-DETR-R101-VD потенциально может достичь более высокой точности при лучших условиях обучения.

С точки зрения практического применения YOLOv8n имеет важные преимущества: размер модели – всего 5.94 МБ, среднее время инференса – 9.6 мс (более 100 FPS). В то же время RT-DETR-R101-VD весит 293.1 МБ и обрабатывает изображения за 158.7 мс (6.3 FPS), что ограничивает её

применение в реальном времени без мощного GPU или оптимизации архитектуры.

Обе модели показали приемлемую точность ($mAP@0.5 > 0.5$), подтвердив применимость в задачах детекции дорожных объектов. YOLOv8n продемонстрировала отличную пригодность для систем реального времени, тогда как RT-DETR-R101-VD подходит для задач, требующих высокой точности, особенно при наличии более мощных вычислительных ресурсов.

Также была выполнена демонстрация работы моделей на разнообразных изображениях с дорожными сценами. Визуальная оценка позволила выявить особенности поведения моделей в разных условиях и наметить направления для дальнейшего повышения их точности и устойчивости.

Таким образом, поставленные цели и задачи были достигнуты. В рамках работы были дообучены и адаптированы к задаче распознавания объектов на дороге нейросетевые модели YOLOv8n и RT-DETR-R101-VD. Полученные результаты пока не обеспечивают уровень качества, необходимый для прямой интеграции в практические системы, однако демонстрируют потенциал и позволяют сделать обоснованный выбор архитектур и методов обучения для дальнейшего развития подобных решений. С учётом последующей доработки и предложенных рекомендаций, используемые модели могут быть практически интегрированы в состав систем компьютерного зрения для автономного вождения и помощи водителю.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Кузнецова, И. О. Возникновение искусственного интеллекта, его преимущества и недостатки / И. О. Кузнецова, Ю. В. Шляпина // Цивилизационные перемены в России : материалы XIII Всероссийской научно-практической конференции, Екатеринбург, 15 мая 2023 года. – Екатеринбург : федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Уральский государственный лесотехнический университет», 2023. – С. 83–87. – EDN ANCEAE.
2. Гафаров, Ф. М. Искусственные нейронные сети и приложения : учебное пособие / Ф. М. Гафаров, А. Ф. Галимянов. – Казань : Изд-во Казан. ун-та, 2018. – 121 с.
3. Антонов, В. Ф. Применение нейронных сетей в задачах искусственного интеллекта / В. Ф. Антонов, Е. А. Поздняков, С. А. Орехов // Современная наука и инновации. – 2021. – № 2(34). – С. 154-160. – DOI 10.37493/2307-910X.2021.2.15. – EDN TSAHSF.
4. Дмитричев, А. С. Нелинейные динамические модели нейронов: обзор / А. С. Дмитричев, Д. В. Касаткин, В. В. Клиньшов, С. Ю. Кириллов [и др.] // Известия вузов. ПНД. – 2018. – № 4. – URL: <https://cyberleninka.ru/article/n/nelineynye-dinamicheskie-modeli-neuronov-obzor> (дата обращения: 06.05.2025).
5. Ростовцев, В. С. Искусственные нейронные сети: учебник / В.С. Ростовцев. – Киров: Изд-во ВятГУ, 2014. – 208 с. Э4743.
6. Круглов, В. В. Искусственные нейронные сети. Теория и практика : учебник / В. В. Круглов, В. В. Борисов. – 2-е изд., стереотип. – Москва : Горячая линия-Телеком, 2002. – 382 с. : ил. – ISBN 5-93517-031-0.
7. Кузнецова, И. О. Принцип работы и архитектура нейронных сетей / И. О. Кузнецова, Д. А. Малютов // Евразийская интеграция: современные тренды и перспективные направления : Материалы VII Международной научно-

практической конференции, Омск, 12 марта 2024 года. – Омск: Омский государственный технический университет, 2024. – С. 106-111. – DOI 10.24412/cl-37031-2024-2-106-111. – EDN IQWDUX.

8. Sharma, S. Activation functions in neural networks / S. Sharma, S. Sharma, A. Athaiya // Towards Data Science. – 2017. – Vol. 6, no. 12. – P. 310–316.

9. Сергеев, А. П. Введение в нейросетевое моделирование : учебное пособие / А. П. Сергеев, Д. А. Тарасов ; под общей редакцией А. П. Сергеева. – Екатеринбург : Изд-во Урал. ун-та, 2017. – 128 с.

10. Степанов, П. П. Искусственные нейронные сети / П. П. Степанов // Молодой ученый. – 2017. – № 4(138). – С. 185-187. – EDN XSCHTZ.

11. Мелерзанов, А. Диагностика меланомы кожи с помощью сверточных нейронных сетей глубокого обучения / А. Мелерзанов, Д. Гаврилов // Врач. – 2018. – Т. 29, № 6. – С. 31–33.

12. Рогаль, А. А. Применение методов глубокого обучения в задаче распознавания изображений / А. А. Рогаль // In Situ. – 2016. – № 6. – С. 13–17.

13. Денисенко, А. А. Глубокое обучение для классификации изображений в различных цветовых системах / А. А. Денисенко // Международный журнал прикладных и фундаментальных исследований. – 2020. – № 8. – С. 42–47.

14. Чичкарев, Е. Использование моделей машинного обучения и сетей глубокого обучения для распознавания рукописных чисел и букв русского и латинского алфавитов / Е. Чичкарев, А. Сергиенко, Е. Балалаева // Scientific Collection «InterConf». Concepts for the development of society's scientific potential (InterConf). – 2021. – Т. 87. – С. 363–380.

15. Нгуен, Т. К. Модель метода распознавания объектов на изображениях с использованием «сверточной нейронной сети – CNN» / Т. К. Нгуен, В. И. Сырямкин, Ч. Х. Т. Нгуен // Современные наукоемкие технологии. – 2020. – № 12-2. – С. 269-280. – DOI 10.17513/snt.38445. – EDN SXHGT.

16. Полковникова, Н. А. Нейросетевые системы автоматического распознавания морских объектов / Н. А. Полковникова // Эксплуатация морского транспорта. – 2020. – № 1(94). – С. 207-218. – DOI 10.34046/aumsuomt94/29. – EDN ZKTKCC.
17. Krizhevsky, A. ImageNet classification with deep convolutional neural networks / A. Krizhevsky, I. Sutskever, G. E. Hinton // Proceedings of the Advances in Neural Information Processing Systems (NIPS). – 2012.
18. LeCun, Y. Backpropagation applied to handwritten zip code recognition / Y. LeCun, B. Boser, J. S. Denker, D. Henderson [и др.] // Neural Computation. – 1989.
19. Liu, W. SSD: Single shot multibox detector / W. Liu, D. Anguelov, D. Erhan, C. Szegedy [и др.] // Proceedings of the European Conference on Computer Vision (ECCV). – 2016.
20. Long, J. Fully convolutional networks for semantic segmentation / J. Long, E. Shelhamer, T. Darrell // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – 2015.
21. Ren, S. Faster R-CNN: Towards real-time object detection with region proposal networks / S. Ren, K. He, R. Girshick, J. Sun // Proceedings of the Advances in Neural Information Processing Systems (NIPS). – 2015.
22. Russakovsky, O. ImageNet large scale visual recognition challenge / O. Russakovsky, J. Deng, H. Su, J. Krause [и др.] // International Journal of Computer Vision (IJCV). – 2015.
23. Simonyan, K. Very deep convolutional networks for large-scale image recognition / K. Simonyan, A. Zisserman // International Conference on Learning Representations (ICLR). – 2015.
24. Szegedy, C. Going deeper with convolutions / C. Szegedy, W. Liu, Y. Jia, P. Sermanet [и др.] // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – 2015.

25. Chen, L.-C. Semantic image segmentation with deep convolutional nets and fully connected CRFs / L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, A. L. Yuille // International Conference on Learning Representations (ICLR). – 2015
26. Yu, F. Multi-scale context aggregation by dilated convolutions / F. Yu, V. Koltun // International Conference on Learning Representations (ICLR). – 2016.
27. Zhao, H. Pyramid scene parsing network / H. Zhao, J. Shi, X. Qi, X. Wang, J. Jia // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – 2017.
28. Захаренко, Г. И. Формирование изображений в системе технического зрения с помощью датчиков // Прикладная математика и информатика: современные исследования в области естественных и технических наук. – 2021. – С. 256-261.
29. Менциев, А. У. Анализ и обучение модели нейронной сети распознавания дорожных знаков / А. У. Менциев, Т. Г. Айгумов, Э. М. Абдулмукминова // Вестник Дагестанского государственного технического университета. Технические науки. – 2023. – № 3 (50). – С. 118–123. – DOI: <https://doi.org/10.21822/2073-6185-2023-50-3-118-123>.
30. Полунин, А. А. Использование аппарата свёрточных нейронных сетей для стегоанализа цифровых изображений / А. А. Полунин, Э. А. Яндашевская // Труды Института системного программирования РАН. – 2020. – Т. 32, № 4. – С. 155-164. – DOI 10.15514/ISPRAS-2020-32(4)-11. – EDN HDNTEXT.
31. Vaswani, A. Attention is all you need / A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin // Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS). – 2017. – P. 5998–6008.
32. Dai, Z. Transformer-XL: Attentive language models beyond a fixed-length context / Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, R. Salakhutdinov // Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL). – 2019.

33. Devlin, J. BERT: Pre-training of deep bidirectional transformers for language understanding / J. Devlin, M.-W. Chang, K. Lee, K. Toutanova // Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT). – 2019.
34. Wu, F. Pay less attention with lightweight and dynamic convolutions / F. Wu, A. Fan, A. Baevski, Y. N. Dauphin, M. Auli // International Conference on Learning Representations (ICLR). – 2019.
35. Yang, Z. XLNet: Generalized autoregressive pretraining for language understanding / Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, Q. V. Le // Advances in Neural Information Processing Systems (NeurIPS). – 2019.
36. Chen, L. SCA-CNN: Spatial and channel-wise attention in convolutional networks for image captioning / L. Chen, H. Zhang, J. Xiao, L. Nie, J. Shao, W. Liu, T.-S. Chua // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – 2017.
37. Dai, J. Deformable convolutional networks / J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, Y. Wei // Proceedings of the IEEE International Conference on Computer Vision (ICCV). – 2017.
38. Hu, J. Gather-excite: Exploiting feature context in convolutional neural networks / J. Hu, L. Shen, S. Albanie, G. Sun, A. Vedaldi // Advances in Neural Information Processing Systems (NeurIPS). – 2018.
39. Parmar, N. Image transformer / N. Parmar, A. Vaswani, J. Uszkoreit, Ł. Kaiser, N. Shazeer, A. Ku, D. Tran // Proceedings of the 35th International Conference on Machine Learning (ICML). – 2018.
40. Ramachandran, P. Stand-alone self-attention in vision models / P. Ramachandran, N. Parmar, A. Vaswani, I. Bello, A. Levskaya, J. Shlens // Advances in Neural Information Processing Systems (NeurIPS). – 2019.
41. Hu, H. Local relation networks for image recognition / H. Hu, Z. Zhang, Z. Xie, S. Lin // Proceedings of the IEEE International Conference on Computer Vision (ICCV). – 2019.

42. Bello, I. Attention augmented convolutional networks / I. Bello, B. Zoph, A. Vaswani, J. Shlens, Q. V. Le // Proceedings of the IEEE International Conference on Computer Vision (ICCV). – 2019.
43. Wang, F. Residual attention network for image classification / F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, X. Tang // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – 2017.
44. Wang, X. Non-local neural networks / X. Wang, R. Girshick, A. Gupta, K. He // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – 2018.
45. Xu, K. Show, attend and tell: Neural image caption generation with visual attention / K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. Zemel, Y. Bengio // Proceedings of the 32nd International Conference on Machine Learning (ICML). – 2015.
46. Yang, Z. Stacked attention networks for image question answering / Z. Yang, X. He, J. Gao, L. Deng, A. Smola // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – 2016.
47. Zhang, H. Self-attention generative adversarial networks / H. Zhang, I. Goodfellow, D. Metaxas, A. Odena // arXiv preprint arXiv:1805.08318. – 2018.
48. Zhao, H. PSANet: Point-wise spatial attention network for scene parsing / H. Zhao, Y. Zhang, S. Liu, J. Shi, C. C. Loy, D. Lin, J. Jia // Proceedings of the European Conference on Computer Vision (ECCV). – 2018.
49. Zhao, H. Exploring self-attention for image recognition / H. Zhao, J. Jia, V. Koltun // Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). – 2020. – P. 10073–10082. – DOI: 10.1109/CVPR42600.2020.01009.
50. Ma, Ch. Why self-attention is natural for sequence-to-sequence problems? A perspective from symmetries / Ch. Ma, L. Ying // Journal of Machine Learning. – 2023. – Vol. 2, No. 3. – P. 194–210. – DOI: 10.4208/jml.221206.

51. Ambartsoumian, A. Self-attention: a better building block for sentiment analysis neural network classifiers / A. Ambartsoumian, F. Popowich // Proceedings of the 9th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis (WASSA@EMNLP 2018). – Brussels, Belgium: Association for Computational Linguistics, 2018. – P. 130–139. – DOI: 10.18653/v1/W18-6219.
52. Goodfellow, I. Deep learning [Электронный ресурс] / I. Goodfellow, Y. Bengio, A. Courville. – Cambridge, MA: MIT Press, 2016. – 775 p. – URL: <https://www.deeplearningbook.org/> (дата обращения: 29.05.2025).
53. Zhou, Z.-H. A brief introduction to weakly supervised learning / Z.-H. Zhou // National Science Review. – 2017. – Vol. 5, № 1. – С. 44–53. – DOI: 10.1093/nsr/nwx103.
54. LeCun, Y. Efficient BackProp / Y. LeCun, L. Bottou, G. B. Orr, K.-R. Müller // Neural Networks: Tricks of the Trade. – Berlin: Springer, 2012. – С. 9–50.
55. Kingma, D. P. Adam: A Method for Stochastic Optimization [Электронный ресурс] / D. P. Kingma, J. Ba // International Conference on Learning Representations (ICLR), 2015. – URL: <https://arxiv.org/abs/1412.6980> (дата обращения: 11.05.2025).
56. Bottou, L. Large-scale machine learning with stochastic gradient descent / L. Bottou // Proceedings of COMPSTAT'2010. – Springer, 2010. – P. 177–186.
57. Duchi, J. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization / J. Duchi, E. Hazan, Y. Singer // Journal of Machine Learning Research. – 2011. – Vol. 12. – P. 2121–2159.
58. Real, E. Regularized Evolution for Image Classifier Architecture Search / E. Real, A. Aggarwal, Y. Huang, Q. V. Le // Proceedings of the AAAI Conference on Artificial Intelligence. – 2019. – Vol. 33, No. 1. – P. 4780–4789.
59. Liu, H. DARTS: Differentiable Architecture Search [Электронный ресурс] / H. Liu, K. Simonyan, Y. Yang // International Conference on Learning Representations (ICLR), 2019. – URL: <https://arxiv.org/abs/1806.09055> (дата обращения: 11.05.2025).

60. Shorten, C. A survey on image data augmentation for deep learning / C. Shorten, T. M. Khoshgoftaar // Journal of Big Data. – 2019. – Vol. 6. – Article 60. – DOI: 10.1186/s40537-019-0197-0.
61. COCO - Common Objects in Context [Электронный ресурс]. – Режим доступа: <https://cocodataset.org>, свободный. – Дата обращения: 10.05.2025.
62. Cubuk, E. D. RandAugment: Practical Automated Data Augmentation with a Reduced Search Space / E. D. Cubuk, B. Zoph, J. Shlens, Q. V. Le // Proc. of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). – 2020. – P. 702–712.
63. Yosinski, J. How transferable are features in deep neural networks? / J. Yosinski, J. Clune, Y. Bengio, H. Lipson // Proc. of Advances in Neural Information Processing Systems (NeurIPS). – 2014. – Vol. 27.
64. GeeksforGeeks. Early Stopping for Regularisation in Deep Learning [Электронный ресурс]. – 2023. – URL: <https://www.geeksforgeeks.org/early-stopping-for-regularisation-in-deep-learning/> (дата обращения: 29.05.2025).
65. Hernández-García, A. Data Augmentation Instead of Explicit Regularization [Электронный ресурс] / A. Hernández-García, P. König. – arXiv preprint arXiv:1806.03852, 2018. – URL: <https://arxiv.org/abs/1806.03852> (дата обращения: 29.05.2025).
66. Li, D. Improved Regularization and Robustness for Fine-tuning in Neural Networks [Электронный ресурс] / D. Li, H. R. Zhang. – arXiv preprint arXiv:2111.04578, 2021. – URL: <https://arxiv.org/abs/2111.04578> (дата обращения: 29.05.2025).
67. Raschka, S. Lecture 10: Regularization Methods for Neural Networks [Электронный ресурс] / S. Raschka. – University of Wisconsin–Madison, 2021. – 25 p. – URL: https://sebastianraschka.com/pdf/lecture-notes/stat453ss21/L10_regularization_slides.pdf (дата обращения: 29.05.2025).
68. Bochkovskiy, A. YOLOv4: Optimal speed and accuracy of object detection [Электронный ресурс] / A. Bochkovskiy, C. Y. Wang, H. Y. M. Liao //

arXiv:2004.10934 [препринт], 2020. – URL: <https://arxiv.org/abs/2004.10934> (дата обращения: 22.05.2025).

69. Kümmerle, J. Vision-based deep learning for autonomous driving: A survey / J. Kümmerle, A. Gern, A. El Sallab [и др.] // IEEE Transactions on Intelligent Transportation Systems. – 2022. – Vol. 23, № 12. – P. 23965–23986. – DOI: 10.1109/TITS.2022.3154422.

70. Wu, C. Y. Long-term feature banks for detailed video understanding / C. Y. Wu, C. Feichtenhofer, H. Fan [и др.] // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – 2019. – P. 284–293.

71. Wang, X. Action recognition with temporal segment networks / X. Wang, Y. Qiao, X. Tang // IEEE Transactions on Pattern Analysis and Machine Intelligence. – 2016. – Vol. 39, № 8. – P. 1533–1545.

72. Ultralytics. Ultralytics YOLO models [Электронный ресурс]. – Режим доступа: <https://github.com/ultralytics/ultralytics>, свободный. – Дата обращения: 17.05.2025.

73. Hugging Face. Transformers documentation [Электронный ресурс]. – Режим доступа: <https://huggingface.co/docs/transformers/>, свободный. – Дата обращения: 17.05.2025.

74. He, H. Learning from imbalanced data / H. He, E. A. Garcia // IEEE Transactions on Knowledge and Data Engineering. – 2009. – Vol. 21, № 9. – P. 1263–1284. – DOI: 10.1109/TKDE.2008.239.

75. Chawla, N. V. SMOTE: Synthetic Minority Over-sampling Technique / N. V. Chawla, K. W. Bowyer, L. O. Hall, W. P. Kegelmeyer // Journal of Artificial Intelligence Research. – 2002. – Vol. 16. – P. 321–357.

76. Lin, T.-Y. Focal Loss for Dense Object Detection / T.-Y. Lin, P. Goyal, R. Girshick, K. He, P. Dollar // Proceedings of the IEEE International Conference on Computer Vision (ICCV). – 2017. – P. 2980–2988.

77. Cui, Y. Class-Balanced Loss Based on Effective Number of Samples / Y. Cui, M. Jia, T.-Y. Lin, Y. Song, S. Belongie // Proceedings of the IEEE/CVF

Conference on Computer Vision and Pattern Recognition (CVPR). – 2019. – P. 9268–9277.

78. Buda, M. A Systematic Study of the Class Imbalance Problem in Convolutional Neural Networks / M. Buda, A. Maki, M. A. Mazurowski // *Neural Networks*. – 2018. – Vol. 106. – P. 249–259.

79. Кузнецов, С. Оценка качества алгоритмов обнаружения объектов на изображениях / С. Кузнецов // *Вестник компьютерных и информационных технологий*. – 2020. – № 3. – С. 12–17.

80. Everingham, M. The PASCAL Visual Object Classes (VOC) Challenge / M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, A. Zisserman // *International Journal of Computer Vision*. – 2010. – Vol. 88, № 2. – P. 303–338. – DOI: 10.1007/s11263-009-0275-4.

81. Szeliski, R. *Computer Vision: Algorithms and Applications* / R. Szeliski. – 2nd ed. – Springer, 2022. – 979 p.

82. Lin, T.-Y. Microsoft COCO: Common Objects in Context / T.-Y. Lin, M. Maire, S. Belongie [et al.] // *ECCV 2014*. – Springer, 2014. – P. 740–755. – DOI: 10.1007/978-3-319-10602-1_48.

83. Padilla, R. A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit / R. Padilla, W. L. Passos, T. L. Dias [et al.] // *Electronics*. – 2021. – Vol. 10, № 3. – P. 279. – DOI: 10.3390/electronics10030279.

84. Zhao, Z.-Q. Object Detection with Deep Learning: A Review / Z.-Q. Zhao, P. Zheng, S.-T. Xu, X. Wu // *IEEE Transactions on Neural Networks and Learning Systems*. – 2019. – Vol. 30, № 11. – P. 3212–3232. – DOI: 10.1109/TNNLS.2018.2876865.

85. Redmon, J. YOLOv3: An Incremental Improvement [Электронный ресурс] / J. Redmon, A. Farhadi // arXiv:1804.02767 [препринт], 2018. – URL: <https://arxiv.org/abs/1804.02767> (дата обращения: 22.05.2025).

86. Caesar, H. nuScenes: A multimodal dataset for autonomous driving / H. Caesar, V. Bankiti, A. H. Lang [и др.] // *Proceedings of the IEEE/CVF Conference on*

Computer Vision and Pattern Recognition. – 2020. – P. 11621–11631. – DOI: 10.1109/CVPR42600.2020.01164.

87. Sun, P. Scalability in Perception for Autonomous Driving: Waymo Open Dataset / P. Sun, H. Kretzschmar, X. Dotiwalla [и др.] // Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. – 2020. – P. 2446–2454. – DOI: 10.1109/CVPR42600.2020.00252.

88. Janai, J. Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art / J. Janai, F. Güney, A. Behl, A. Geiger // Foundations and Trends® in Computer Graphics and Vision. – 2020. – Vol. 12, no. 1–3. – P. 1–308. – DOI: 10.1561/06000000079.

89. NVIDIA. Deep Learning Inference Performance: NVIDIA TensorRT and Deep-Stream SDK. – 2021. – [Electronic resource] – URL: <https://developer.nvidia.com/tensorrt> (дата обращения: 23.05.2025).

90. Intel Corporation. Edge AI Performance Requirements for Computer Vision in Autonomous Vehicles [Электронный ресурс]. – White Paper. – 2020. – URL: <https://www.intel.com/content/www/us/en/automotive/edge-ai-autonomous-whitepaper.html> (дата обращения: 23.05.2025).

91. Redmon, J. You Only Look Once: Unified, Real-Time Object Detection / J. Redmon, S. Divvala, R. Girshick, A. Farhadi // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – 2016. – P. 779–788. – DOI: 10.1109/CVPR.2016.91. – URL: <https://pjreddie.com/media/files/papers/yolo.pdf> (дата обращения: 17.05.2025).

92. Tripathi, A. Comparative analysis of object detection algorithms for autonomous vehicle applications / A. Tripathi, R. Balasubramanian // Multimedia Tools and Applications. – 2022. – Vol. 81, Iss. 21. – P. 29861–29881. – DOI: <https://doi.org/10.1007/s11042-022-12447-5>.

93. Raj, K. A Review on Deep Learning approaches for Object Detection in Self-Driving Cars / K. Raj, P. Rajendran // NeuroQuantology. – 2023. – Vol. 21, No.

8. – P. 1704–1711. – URL: <https://www.neuroquantology.com/article.php?id=9862> (дата обращения: 23.05.2025).

94. Ren, Z. MST-YOLO: Small Object Detection Model for Autonomous Driving / Z. Ren, H. Zhang, Z. Li // *Sensors*. – 2024. – Vol. 24, No. 22. – Article 7347. – DOI: 10.3390/s24227347.

95. Ren, Z. Improved YOLOv5 Network for Real-Time Object Detection in Vehicle-Mounted Camera Capture Scenarios / Z. Ren, H. Zhang, Z. Li // *Sensors*. – 2023. – Vol. 23, No. 10. – Article 4589. – DOI: 10.3390/s23104589.

96. Fahad, I. A. Automatic Vehicle Detection using DETR: A Transformer-Based Approach for Navigating Treacherous Roads [Электронный ресурс] / I. A. Fahad, A. I. H. Arian, N. S. Ahmed, M. Hasan // *arXiv:2502.17843* [препринт], 2025. – URL: <https://arxiv.org/abs/2502.17843> (дата обращения: 29.05.2025).

97. Wang, S. RT-DETRv3: Real-time End-to-End Object Detection with Hierarchical Dense Positive Supervision [Электронный ресурс] / S. Wang, C. Xia, F. Lv, Y. Shi // *arXiv:2409.08475* [препринт], 2024. – URL: <https://arxiv.org/abs/2409.08475> (дата обращения: 29.05.2025).

98. Chen, Q. LW-DETR: A Transformer Replacement to YOLO for Real-Time Detection [Электронный ресурс] / Q. Chen, X. Su, X. Zhang, J. Wang, J. Chen, Y. Shen, C. Han, Z. Chen, W. Xu, F. Li, S. Zhang, K. Yao, E. Ding, G. Zhang, J. Wang // *arXiv:2406.03459* [препринт], 2024. – URL: <https://arxiv.org/abs/2406.03459> (дата обращения: 29.05.2025).

99. Zhao, H. Improved Object Detection Method for Autonomous Driving Based on DETR / H. Zhao, S. Zhang, X. Peng, Z. Lu, G. Li // *Frontiers in Neurorobotics*. – 2025. – Vol. 18. – Article 1484276. – DOI: 10.3389/fnbot.2024.1484276.

100. Dong, X. CTAFFNet: CNN–Transformer Adaptive Feature Fusion Object Detection Algorithm for Complex Traffic Scenarios / X. Dong, P. Shi, T. Liang, A. Yang // *Transportation Research Record*. – 2024. – Vol. 2678, No. 2. – P. 123–135. – DOI: 10.1177/03611981241258753.

101. Lu, K. An Object Detection Algorithm Combining Self-Attention and YOLOv4 in Traffic Scene / K. Lu, F. Zhao, X. Xu, Y. Zhang // PLoS ONE. – 2023. – Vol. 18, No. 5. – e0285654. – DOI: 10.1371/journal.pone.0285654.
102. Li, Y. Improved Vehicle Object Detection Algorithm Based on Swin-YOLOv5s / Y. Li, H. Zhang, X. Chen // Processes. – 2023. – Vol. 11, No. 3. – P. 925. – DOI: 10.3390/pr11030925.
103. РБК Тренды. Формула без пилота: кто в России создает автономный гоночный болид [Электронный ресурс] // РБК Тренды. – URL: <https://trends.rbc.ru/trends/industry/60cb026c9a7947520890aee7> (дата обращения: 09.05.2025).
104. Минниханов, Р. Н. Распознавание сигналов светофора на основе сверточных нейросетевых моделей / Р. Н. Минниханов, В. Скибин, Н. Г. Талипов, А. С. Катасев // Автоматизация процессов управления. – 2024. – № 1(75). – С. 49–57. – DOI: 10.35752/1991-2927_2024_1_75_49. – EDN DECDRM.
105. Ляшева, М. М. Обнаружение пешеходов на изображениях на основе модели глубокого обучения / М. М. Ляшева, С. А. Ляшева, М. П. Шлеймович // Электроника, фотоника и киберфизические системы. – 2023. – Т. 3, № 4. – С. 7–18. – EDN BVGUTG.
106. Deng, J. RT-DETR: Real-Time Detection Transformer [Электронный ресурс] / J. Deng, Z. Yu, T. Lu, R. Liao, J. Zhou, J. Zhao, Z. Li // arXiv:2304.08069 [препринт], 2023. – URL: <https://arxiv.org/abs/2304.08069> (дата обращения: 17.05.2025).
107. Peking University. RT-DETR with ResNet-101-VD backbone, pretrained on COCO and Object365 [Электронный ресурс]. – Режим доступа: https://huggingface.co/PekingU/rtdetr_r101vd_coco_o365 (дата обращения: 23.05.2025).
108. BDD100K: Berkeley DeepDrive Dataset [Электронный ресурс] // Berkeley Artificial Intelligence Research. – URL: <https://bdd-data.berkeley.edu> (дата обращения: 18.05.2025).

109. Solesensei_bdd100k [Электронный ресурс] // Kaggle. – URL: https://www.kaggle.com/datasets/solesensei/solesensei_bdd100k (дата обращения: 18.05.2025).
110. Fedutinov, K. A. Machine Learning in Decision Support Problems in Nature Conservation Management [Электронный ресурс] / К. А. Fedutinov // Inzhenerny Vestnik Dona, 2022, № 9. – URL: <https://ivdon.ru/ru/magazine/archive/n9y2021/7186> (дата обращения: 30.05.2025). – (In Russ).
111. Jupyter Notebook [Электронный ресурс]. – Режим доступа: <https://jupyter.org/> (дата обращения: 17.05.2025).
112. YOLOv8n, обученная на BDD100K [Электронный ресурс]. URL: <https://github.com/WangYangfan/yolov8> (дата обращения: 03.06.2025).
113. YOLOv5s, обученная на BDD100K [Электронный ресурс]. URL: https://github.com/williamhyin/yolov5s_bdd100k. (дата обращения: 03.06.2025).
114. YOLOv8n, обученная на пользовательском датасете для обнаружения дорожных ям [Электронный ресурс]. URL: <https://learnopencv.com/train-yolov8-on-custom-dataset/> (дата обращения: 03.06.2025).
115. YOLOv8n, обученная на датасете JUIVCDv1 для обнаружения транспортных средств [Электронный ресурс]. URL: <https://github.com/Akashkg03/VEHICLE-OBJECT-DETECTION-USING-YOLOv8n> (дата обращения: 03.06.2025).
116. СТУ 7.5–07–2021 Система менеджмента качества. Общие требования к построению, изложению и оформлению документов учебной деятельности. Красноярск: СФУ, 2021. – С. 61.

ПРИЛОЖЕНИЕ А

Отчет о проверке на заимствование

Отчет о проверке

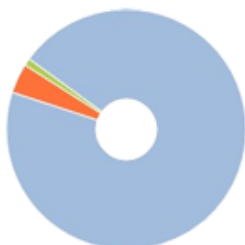
Автор: Паранина Дарья Михайловна

Название документа: Магистерская диссертация ПаранинаДМ КИ23-01-13М

Проверяющий: Захаров Павел Алексеевич

Организация: Федеральное государственное автономное образовательное учреждение высшего образования «Сибирский федеральный университет»

РЕЗУЛЬТАТЫ ПРОВЕРКИ



Совпадения:
4,49%



Оригинальность:
94,73%



Цитирования:
0,78%



Самоцитирования:
0%



«Совпадения», «Цитирования», «Самоцитирования», «Оригинальность» являются отдельными показателями, отображаются в процентах и в сумме дают 100%, что соответствует проверенному тексту документа.

Проверено: 83,9% текста документа, исключено из проверки: 16,1% текста документа. Разделы, отключенные пользователем: Библиография

- **Совпадения** — фрагменты проверяемого текста, полностью или частично сходные с найденными источниками, за исключением фрагментов, которые система отнесла к цитированию или самоцитированию. Показатель «Совпадения» — это доля фрагментов проверяемого текста, отнесенных к совпадениям, в общем объеме текста.
- **Самоцитирования** — фрагменты проверяемого текста, совпадающие или почти совпадающие с фрагментом текста источника, автором или соавтором которого является автор проверяемого документа. Показатель «Самоцитирования» — это доля фрагментов текста, отнесенных к самоцитированию, в общем объеме текста.
- **Цитирования** — фрагменты проверяемого текста, которые не являются авторскими, но которые система отнесла к корректно оформленным. К цитированиям относятся также шаблонные фразы; библиография; фрагменты текста, найденные модулем поиска «СПС Гарант: нормативно-правовая документация». Показатель «Цитирования» — это доля фрагментов проверяемого текста, отнесенных к цитированию, в общем объеме текста.
- **Текстовое пересечение** — фрагмент текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника.
- **Источник** — документ, проиндексированный в системе и содержащийся в модуле поиска, по которому проводится проверка.
- **Оригинальный текст** — фрагменты проверяемого текста, не обнаруженные ни в одном источнике и не отмеченные ни одним из модулей поиска. Показатель «Оригинальность» — это доля фрагментов проверяемого текста, отнесенных к оригинальному тексту, в общем объеме текста.

Обращаем Ваше внимание, что система находит текстовые совпадения проверяемого документа с проиндексированными в системе источниками. При этом система является вспомогательным инструментом, определение корректности и правомерности совпадений или цитирований, а также авторства текстовых фрагментов проверяемого документа остается в компетенции проверяющего.

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

Номер документа: 396273

Тип документа: Магистерская диссертация

Дата проверки: 04.06.2025 22:57:19

Дата корректировки: Нет

Количество страниц: 110

Символов в тексте: 160449

Слов в тексте: 19697

Число предложений: 4281

Рисунок А.1 - Отчет о проверке на заимствование
96

ПРИЛОЖЕНИЕ Б

Преобразование классов и очистка данных BDD100K

```
import json
import os
import shutil

# Выбор split: 'train' или 'val'
split = 'train'

input_path = fr"C:\Users\odara\Downloads\data\labels\bdd100k_labels_images_{split}.json"
output_dir = r"C:\Users\odara\Downloads\data\labels_new"
os.makedirs(output_dir, exist_ok=True)
output_path = os.path.join(output_dir, f"bdd100k_labels_images_{split}.json")

# Классы для удаления и объединения
delete_classes = {'train', 'lane', 'drivable area'}
merge_classes = {'bike', 'motor'}
new_merged_class = 'two_wheeler'

# Загрузка аннотаций
with open(input_path, 'r', encoding='utf-8') as f:
    data = json.load(f)

# Обработка аннотаций
processed_data = []
for item in data:
    new_labels = []
    for label in item.get("labels", []):
        category = label.get("category")

        if category in delete_classes:
            continue # Пропустить нежелательные классы

        if category in merge_classes:
            label["category"] = new_merged_class

    new_labels.append(label)

    # Добавить только если остались аннотации
    if new_labels:
        item["labels"] = new_labels
        processed_data.append(item)

# Сохранение
with open(output_path, 'w', encoding='utf-8') as f:
    json.dump(processed_data, f, ensure_ascii=False, indent=2)

print(f"Готово: сохранено {len(processed_data)} аннотированных изображений в {output_path}")
```

Готово: сохранено 69863 аннотированных изображений в C:\Users\odara\Downloads\data\labels_new\bdd100k_labels_images_train.json

Готово: сохранено 10000 аннотированных изображений в C:\Users\odara\Downloads\data\labels_new\bdd100k_labels_images_val.json

Рисунок Б.1 – Результат преобразования классов в аннотациях BDD100K

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б

```
# Очистка данных
# Параметры

split = 'train' # или 'val'
images_dir = fr"C:\Users\odara\Downloads\data\{split}\images"
annotations_path = fr"C:\Users\odara\Downloads\data\labels_new\bdd100k_labels_images_{split}.json"
bad_images_dir = fr"C:\Users\odara\Downloads\data\bad_samples\images"
bad_labels_dir = fr"C:\Users\odara\Downloads\data\bad_samples\labels"
os.makedirs(bad_images_dir, exist_ok=True)
os.makedirs(bad_labels_dir, exist_ok=True)

# Настройка допустимых классов
valid_classes = {
    "person", "car", "bus", "truck", "traffic sign", "traffic light",
    "two_wheeler", "rider"
}

# Загрузка аннотаций
with open(annotations_path, 'r', encoding='utf-8') as f:
    annotations = json.load(f)

image_names_in_annot = set()
seen_images = set()
valid_annotations = []
bad_ann_count = 0
missing_annot_count = 0
duplicate_ann_count = 0

for entry in annotations:
    image_name = entry.get("name")
    image_path = os.path.join(images_dir, image_name)

    # Проверка на дубликат
    if image_name in seen_images:
        reason = "duplicate_image_name"
        duplicate_ann_count += 1
    else:
        seen_images.add(image_name)
        # Проверка поля labels
        labels = entry.get("labels")
        if labels is None:
            reason = "no_labels_field"
        elif not labels:
            reason = "empty_labels"
        else:
            valid = True
            for label in labels:
                cat = label.get("category")
                box = label.get("box2d")
                if cat not in valid_classes or not box:
                    valid = False
                    reason = f"invalid_label_or_box"
                    break
            if box["x1"] >= box["x2"] or box["y1"] >= box["y2"]:
                valid = False
                reason = "invalid_box_coordinates"
                break
```

ОКОНЧАНИЕ ПРИЛОЖЕНИЯ Б

```
        if valid:
            valid_annotations.append(entry)
            image_names_in_annot.add(image_name)
            continue # всё прошло, пропускаем к следующему

    if os.path.exists(image_path):
        shutil.move(image_path, os.path.join(bad_images_dir, image_name))
    ann_output_path = os.path.join(bad_labels_dir, image_name.replace('.jpg',
'.json'))
    with open(ann_output_path, 'w', encoding='utf-8') as f:
        json.dump(entry, f, ensure_ascii=False, indent=2)
    bad_ann_count += 1

# Проверка изображений без аннотаций
for fname in os.listdir(images_dir):
    if not fname.endswith(".jpg"):
        continue
    if fname not in image_names_in_annot:
        shutil.move(os.path.join(images_dir, fname), os.path.join(bad_images_dir, fname))
    missing_annot_count += 1

# Сохраняем очищенные аннотации
with open(annotations_path, 'w', encoding='utf-8') as f:
    json.dump(valid_annotations, f, ensure_ascii=False, indent=2)

# Отчет

print("Очистка завершена.")
print(f"Удалено изображений без аннотаций: {missing_annot_count}")
print(f"Удалено аннотаций с ошибками (и соответствующих изображений): {bad_ann_count}")
print(f"Найдено и удалено дубликатов изображений: {duplicate_ann_count}")
```

```
Очистка завершена.
Удалено изображений без аннотаций: 137
Удалено аннотаций с ошибками (и соответствующих изображений): 0
Найдено и удалено дубликатов изображений: 0
```

Рисунок Б.2 - Результат очистки тренировочных данных

ПРИЛОЖЕНИЕ В

Oversampling и undersampling тренировочных данных

```
# oversampling

import os
import json
import shutil

input_json_path = r"C:\Users\odara\Downloads\data\labels_new\bdd100k_labels_images_train.json"
output_json_path = r"C:\Users\odara\Downloads\data\labels_new\bdd100k_labels_images_train_oversampled.json"
input_images_dir = r"C:\Users\odara\Downloads\data\train\images"
output_images_dir = r"C:\Users\odara\Downloads\data\train_new\images"

# Целевые классы для оверсэмплинга
target_categories = {"rider", "two_wheeler", "bus"}

# Загрузка аннотаций
with open(input_json_path, 'r') as f:
    annotations = json.load(f)

# Готовим директорию вывода
os.makedirs(output_images_dir, exist_ok=True)

# Новые аннотации
new_annotations = []

for item in annotations:
    name = item["name"]
    labels = item.get("labels", [])

    # Добавляем оригинал
    src_path = os.path.join(input_images_dir, name)
    dst_path = os.path.join(output_images_dir, name)
    if os.path.exists(src_path):
        shutil.copy2(src_path, dst_path)
        new_annotations.append(item)

    # Проверка на нужные классы
    has_target = any(label.get("category") in target_categories for label in labels)

    if has_target:
        # Создаем копию изображения
        name_copy = name.replace(".jpg", "_copy.jpg")
        dst_copy_path = os.path.join(output_images_dir, name_copy)
        shutil.copy2(src_path, dst_copy_path)

        # Создаем копию аннотации с новым именем
        item_copy = json.loads(json.dumps(item)) # глубокая копия
        item_copy["name"] = name_copy
        new_annotations.append(item_copy)
```

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ В

```
# Сохраняем новые аннотации
with open(output_json_path, 'w') as f:
    json.dump(new_annotations, f, indent=2)
print(f"Оверсэмплинг завершён. Сохранено {len(new_annotations)} аннотаций.")
```

Оверсэмплинг завершён. Сохранено 84047 аннотаций.

Рисунок В.1 – Результат oversampling'a

```
# undersampling example
```

```
import json
import os
import shutil
from tqdm import tqdm

images_dir = r"C:\Users\odara\Downloads\data\train_new\images"
labels_path = r"C:\Users\odara\Downloads\data\labels_new\bdd100k_labels_images_train_oversampled.json"

deleted_images_dir = r"C:\Users\odara\Downloads\data\deleted_car_samples\images"
deleted_labels_dir = r"C:\Users\odara\Downloads\data\deleted_car_samples\labels"

os.makedirs(deleted_images_dir, exist_ok=True)
os.makedirs(deleted_labels_dir, exist_ok=True)

with open(labels_path, "r") as f:
    annotations = json.load(f)

target_class = "car"
target_class_threshold = 350_000
protected_classes = {"rider", "two_wheeler", "bus"} # Добавляем сюда

def count_objects(data, cls):
    total = 0
    for ann in data:
        cats = [obj["category"] for obj in ann.get("labels", [])]
        total += cats.count(cls)
    return total

total_cars = count_objects(annotations, target_class)
print(f"Всего объектов '{target_class}': {total_cars}")

filtered_annotations = []
deleted_count = 0

for ann in tqdm(annotations, desc="Processing images"):
    cats = [obj["category"] for obj in ann.get("labels", [])]
    unique_cats = set(cats)
    car_count = cats.count(target_class)

    # Новое условие: если есть хоть один "protected" класс — не удаляем
    if protected_classes.intersection(unique_cats):
        filtered_annotations.append(ann)
        continue
```

ОКОНЧАНИЕ ПРИЛОЖЕНИЯ В

```
# Условие удаления: есть только "car" и не меньше порога
if unique_cats == {target_class}:
    if total_cars - car_count < target_class_threshold:
        filtered_annotations.append(ann)
        continue
    # Копируем изображение
    src_img_path = os.path.join(images_dir, ann["name"])
    dst_img_path = os.path.join(deleted_images_dir, ann["name"])
    if os.path.exists(src_img_path):
        shutil.copy2(src_img_path, dst_img_path)
        os.remove(src_img_path)

    # Сохраняем аннотацию отдельным JSON
    label_filename = os.path.splitext(ann["name"])[0] + ".json"
    dst_label_path = os.path.join(deleted_labels_dir, label_filename)
    with open(dst_label_path, "w") as f_label:
        json.dump(ann, f_label, indent=2)

    total_cars -= car_count
    deleted_count += 1
else:
    filtered_annotations.append(ann)

print(f"Удалено изображений: {deleted_count}")
print(f"Осталось автомобилей '{target_class}': {total_cars}")

# Сохраняем обновлённые аннотации в новый файл, не затирая исходный
output_labels_path = r"C:\Users\odara\Downloads\data\labels_new\bdd100k_labels_images_train_undersampled_1.json"
with open(output_labels_path, "w") as f_out:
    json.dump(filtered_annotations, f_out, indent=2)
```

Всего объектов 'car': 856076

Processing images: 100% | 84047/84047 [00:20<00:00, 4170.0 images/s]

Удалено изображений: 4322

Осталось автомобилей 'car': 810444

Рисунок В.2 – Результат выполнения кода одного из четырех этапов under-sampling'a класса car

ПРИЛОЖЕНИЕ Г

Преобразование тренировочных данных BDD100K в форматы YOLO и COCO

Преобразование в YOLO -формат

```
import json
import os
```

Пути

```
input_json = r"C:\Users\odara\Downloads\data\labels_new\bdd100k_labels_images_train_undersampled_4.json"
output_dir = r"C:\Users\odara\Downloads\data\train_new\labels"
```

Классы

```
class_map = {
    "car": 0,
    "traffic sign": 1,
    "traffic light": 2,
    "person": 3,
    "truck": 4,
    "bus": 5,
    "two_wheeler": 6,
    "rider": 7}
```

Разрешение изображений BDD100K

```
img_width = 1280
img_height = 720
```

Убедимся, что папка существует

```
os.makedirs(output_dir, exist_ok=True)
```

Загрузка аннотаций

```
with open(input_json, "r") as f:
    data = json.load(f)
```

Обработка

```
for item in data:
    image_name = item["name"]
    label_filename = os.path.splitext(image_name)[0] + ".txt"
    label_path = os.path.join(output_dir, label_filename)
    lines = []
    for label in item.get("labels", []):
        category = label.get("category")
        box2d = label.get("box2d")

        if category in class_map and box2d:
            x1 = box2d["x1"]
            y1 = box2d["y1"]
            x2 = box2d["x2"]
            y2 = box2d["y2"]
```

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Г

```
# YOLO формат – нормализованные координаты
x_center = ((x1 + x2) / 2) / img_width
y_center = ((y1 + y2) / 2) / img_height
width = (x2 - x1) / img_width
height = (y2 - y1) / img_height
class_id = class_map[category]
line = f"{class_id} {x_center:.6f} {y_center:.6f} {width:.6f}
{height:.6f}"
lines.append(line)

with open(label_path, "w") as f:
    f.write("\n".join(lines))

# Преобразование в СОСО-формат

import json
import os
from tqdm import tqdm

# Пути
bdd_json_path = rf"C:\Users\odara\Downloads\data\labels_new\bdd100k_labels_images_train_undersampled_4.json"
images_dir = rf"C:\Users\odara\Downloads\data\train_new\images"
output_json = rf"C:\Users\odara\Downloads\data\rtddetr_labels\instances_train.json"

# Размер изображений BDD100K
img_width = 1280
img_height = 720

# Классы, которые используем
class_map = {
    "car": 0,
    "traffic sign": 1,
    "traffic light": 2,
    "person": 3,
    "truck": 4,
    "bus": 5,
    "two_wheeler": 6,
    "rider": 7
}

# COCO-структура
coco_output = {
    "images": [],
    "annotations": [],
    "categories": [{"id": v, "name": k} for k, v in class_map.items()]
}

# Счетчики
image_id = 0
annotation_id = 0

# Загружаем BDD JSON
with open(bdd_json_path, "r") as f:
    bdd_data = json.load(f)
```

ОКОНЧАНИЕ ПРИЛОЖЕНИЯ Г

```
for item in tqdm(bdd_data):
    file_name = item["name"]
    image_path = os.path.join(images_dir, file_name)
    if not os.path.exists(image_path):
        continue # Пропускаем, если изображения нет
    coco_output["images"].append({
        "id": image_id,
        "file_name": file_name,
        "width": img_width,
        "height": img_height
    })

    for label in item.get("labels", []):
        category = label.get("category")
        if category not in class_map or "box2d" not in label:
            continue

        box = label["box2d"]
        x1 = box["x1"]
        y1 = box["y1"]
        x2 = box["x2"]
        y2 = box["y2"]
        w = x2 - x1
        h = y2 - y1

        coco_output["annotations"].append({
            "id": annotation_id,
            "image_id": image_id,
            "category_id": class_map[category],
            "bbox": [x1, y1, w, h],
            "area": w * h,
            "iscrowd": 0
        })
        annotation_id += 1

    image_id += 1

# Сохраняем COCO JSON
os.makedirs(os.path.dirname(output_json), exist_ok=True)
with open(output_json, "w") as f:
    json.dump(coco_output, f)

print(f"Сохранено в: {output_json}")
```

ПРИЛОЖЕНИЕ Д

Создание кастомного датасета для дообучения RT-DETR

```
import json
from PIL import Image
import torch
from torch.utils.data import Dataset

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

class CustomCocoDataset(Dataset):
    def __init__(self, images_dir, annotations_file, processor):
        self.images_dir = images_dir
        self.processor = processor

        # Загружаем аннотации из COCO JSON
        with open(annotations_file, 'r') as f:
            coco = json.load(f)

        # Строим словарь: image_id информация об изображении
        self.image_id_to_info = {img['id']: img for img in coco['images']}

        # Группируем аннотации по image_id
        self.annotations_per_image = {}
        for ann in coco['annotations']:
            img_id = ann['image_id']
            if img_id not in self.annotations_per_image:
                self.annotations_per_image[img_id] = []
            self.annotations_per_image[img_id].append(ann)

        # Создаём список всех image_id для итерирования
        self.image_ids = list(self.image_id_to_info.keys())

    def __len__(self):
        return len(self.image_ids)

    def __getitem__(self, idx):
        image_id = self.image_ids[idx]
        img_info = self.image_id_to_info[image_id]

        # Загружаем изображение
        img_path = f"{self.images_dir}/{img_info['file_name']}"
        image = Image.open(img_path).convert("RGB")
```

ОКОНЧАНИЕ ПРИЛОЖЕНИЯ Д

```
# Получаем аннотации
anns = self.annotations_per_image.get(image_id, [])
annotations = []
for ann in anns:
    bbox = ann["bbox"] # [x, y, width, height]
    area = ann.get("area", bbox[2] * bbox[3]) # если нет 'area', считаем
сами
    iscrowd = ann.get("iscrowd", 0)
    annotations.append({
        "bbox": bbox,
        "category_id": ann["category_id"],
        "area": area,
        "iscrowd": iscrowd,
    })
    annotation_dict = {
        "image_id": image_id,
        "annotations": annotations
    }

    encoding = self.processor(images=image, annotations=annotation_dict, re-
turn_tensors="pt")

    pixel_values = encoding["pixel_values"].squeeze(0).to(device)

    # labels = encoding["labels"][0]
    labels = {
        k: v.to(device) if isinstance(v, torch.Tensor) else v
        for k, v in encoding["labels"][0].items()
    }
    return {
        "pixel_values": pixel_values,
        "labels": labels,
    }

# collate_fn для DataLoader и Trainer

def collate_fn(batch):
    pixel_values = torch.stack([item["pixel_values"] for item in batch])
    labels = [item["labels"] for item in batch] # список словарей с разным
количеством объектов
    return {"pixel_values": pixel_values, "labels": labels}

train_dataset = CustomCocoDataset(
    images_dir=r"C:\Users\odara\Downloads\data\train_new\images",
    annotations_file=r"C:\Users\odara\Downloads\data\rtddetr_labels\in-
stances_train.json",
    processor=processor
)

val_dataset = CustomCocoDataset(
    images_dir=r"C:\Users\odara\Downloads\data\val\images",
    annotations_file=r"C:\Users\odara\Downloads\data\rtddetr_labels\in-
stances_val.json",
    processor=processor)
```