МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ**

**Н. Г. ЧЕРНЫШЕВСКОГО»**

Кафедра информатики и программирования

**Разработка мобильного приложения для распознавания маршрутных номеров с использованием нейронных сетей**

**КУРСОВАЯ РАБОТА**

студента 3 курса 341 группы

Протасова Ивана

направления 02.03.03 Математическое обеспечение и администрирование

информационных систем

факультета компьютерных наук и информационных технологий

Научный руководитель \_\_\_\_\_\_\_\_\_\_ Е.Е. Лапшева

Старший преподаватель

Зав. кафедрой

к.ф.-м.н., доцент \_\_\_\_\_\_\_\_\_\_ М.В. Огнева

**Саратов 2025**

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc199311647)

[1. Обзор существующих решений 6](#_Toc199311648)

[2. Обзор и выбор технологий для реализации приложения 7](#_Toc199311649)

[2.1. Языки программирования 7](#_Toc199311650)

[2.2. Библиотеки и фреймворки 8](#_Toc199311651)

[2.3. Инструменты мобильной разработки 9](#_Toc199311652)

[2.4. Инструменты тестирования 10](#_Toc199311653)

[2.5. Инструмент разметки датасета 11](#_Toc199311654)

[3. Описание предметной области 11](#_Toc199311655)

[3.1. Компьютерное зрение 11](#_Toc199311656)

[3.1.1. Задача распознавания объектов 12](#_Toc199311657)

[3.1.2. Задача распознавания текста 15](#_Toc199311658)

[3.2. Свёрточные нейронные сети 16](#_Toc199311659)

[3.3. Архитектура свёрточной нейронной сети 18](#_Toc199311660)

[3.4. Обучение свёрточной нейронной сети 20](#_Toc199311661)

[3.6. Работа сети после завершения обучения 22](#_Toc199311662)

[4. Практическая часть 22](#_Toc199311663)

[4.1. Подготовка датасета 22](#_Toc199311664)

[4.2. Архитектура нейронной сети 25](#_Toc199311665)

[4.2.1. Общий извлекатель признаков 25](#_Toc199311666)

[4.2.2. Классификатор типа транспорта 27](#_Toc199311667)

[4.2.3. Детекция bounding box 27](#_Toc199311668)

[4.2.4. Модуль распознавания текста 28](#_Toc199311669)

[4.2.5. Прямой проход 29](#_Toc199311670)

[4.2.6. Функции потерь и постобработка 30](#_Toc199311671)

[4.3. Обучение многозадачной модели 32](#_Toc199311672)

[4.4. Экспорт модели 35](#_Toc199311673)

[4.5. Реализация android-приложения 37](#_Toc199311674)

[4.5.1. Проектирование пользовательского интерфейса 37](#_Toc199311675)

[4.5.2. Программная реализация 38](#_Toc199311676)

[4.6 Тестирование и оценка эффективности системы 41](#_Toc199311677)

[ЗАКЛЮЧЕНИЕ 43](#_Toc199311678)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 45](#_Toc199311679)

[ПРИЛОЖЕНИЕ А – Архитектура модели 47](#_Toc199311680)

[ПРИЛОЖЕНИЕ Б – Обучение модели 53](#_Toc199311681)

[ПРИЛОЖЕНИЕ В – Загрузка датасета 57](#_Toc199311682)

# ВВЕДЕНИЕ

**Актуальность** данной темы заключается в необходимости создания решений для облегчения доступа слабовидящих людей к общественному транспорту. Существующие методы, такие как звуковые оповещения и дисплеи высокой контрастности, не всегда эффективны. Использование методов компьютерного зрения позволяет предложить современное и персонализированное решение. Несмотря на успехи в данной области, включая алгоритмы OCR (optical character recognition) и технологии обработки изображений, специализированных решений для распознавания номеров автобусов в сложных условиях пока недостаточно. Важным аспектом данной работы является создание собственного набора данных и разработка специализированной нейронной сети, что позволит учесть специфику задачи и улучшить точность и надежность распознавания.

**Цель курсовой работы:** разработать мобильное приложение с интегрированной нейронной сетью, предназначенное для распознавания вида общественного транспорта и его номера в условиях городской среды.

**Задачи курсовой работы:**

1. Изучить теоретические основы компьютерного зрения и свёрточных нейронных сетей, необходимых для решения задачи распознавания объектов и текста.
2. Проанализировать существующие программные решения и выбрать наиболее подходящие технологии и инструменты для разработки мобильного приложения и обучения нейросети.
3. Подготовить датасет, провести обучение модели нейронной сети и выполнить её оптимизацию для мобильных устройств.
4. Разработать Android-приложение с использованием выбранных инструментов и интегрировать в неё обученную модель.
5. Провести тестирование и оценку эффективности работы разработанной системы в приближённых к реальным условиях.

# ****1. Обзор существующих решений****

На сегодняшний день существует множество решений, направленных на распознавание текста и объектов с помощью компьютерного зрения. Однако большинство из них ориентированы на другие задачи или имеют ограничения, которые делают их недостаточно эффективными для распознавания номеров автобусов в условиях городской среды.

**Системы автоматического распознавания номеров (ANPR)**  
широко используются для распознавания автомобильных номеров. Они основаны на алгоритмах компьютерного зрения, таких как YOLO или SSD для детекции объекта и OCR для извлечения текста. Эти системы демонстрируют высокую точность в контролируемых условиях (фиксированные камеры, стабильное освещение). Однако их применение ограничено, поскольку они не предназначены для мобильных устройств, работы в сложных условиях освещения или распознавания номеров на движущемся транспорте. [1]

**Seeing AI** от Microsoft способен распознавать текст и объекты, предоставляя голосовую обратную связь. Тем не менее, это приложение являются универсальным и не адаптировано для задач, связанных с идентификацией номеров автобусов. Это приводит к снижению эффективности и удобства использования в подобных сценариях. [2]

**Инфраструктурные решения в транспорте некоторых** городов, такие как внедрение систем озвучивания маршрутов или дисплеев высокой контрастности на автобусах, требуют значительных финансовых вложений и модернизации инфраструктуры, что делает их недоступными для большинства регионов. Более того, их эффективность ограничена, так как они не учитывают индивидуальные потребности слабовидящих пользователей.

**Таким образом**, несмотря на наличие аналогов, ни одно из существующих решений не удовлетворяет всем требованиям задачи распознавания номеров автобусов для слабовидящих пользователей.

# ****2.**** Обзор и выбор технологий для реализации приложения

## 2.1. Языки программирования

**Python** — это высокоуровневый язык программирования, который широко используется в области машинного обучения и искусственного интеллекта. Он имеет богатую экосистему библиотек для работы с данными, таких как NumPy, Pandas, PyTorch, TensorFlow, а также поддерживает удобные фреймворки для разработки и обучения нейронных сетей.

* **Почему выбран**: Python был выбран для разработки нейронной сети в связи с его высокой продуктивностью в области искусственного интеллекта и машинного обучения. Он обладает простой и понятной синтаксисической структурой, что позволяет быстро прототипировать и разрабатывать модели, а также активно поддерживается научным сообществом.
* **Альтернативы**: C++ может быть более эффективным по времени выполнения и имеет большой контроль над низкоуровневыми операциями, что важно для разработки высокоскоростных систем, однако не имеет таких удобных и мощных библиотек для машинного обучения, что делает процесс разработки более трудозатратным. [3]

**Kotlin** — это современный язык программирования, который используется для создания приложений под Android. Kotlin полностью совместим с Java, но обладает более современным синтаксисом и функциональными возможностями.

* **Почему выбран**: Kotlin выбран для создания Android-приложения, поскольку это официальный язык разработки для Android, поддерживаемый Google. Он предоставляет безопасную работу с нулевыми значениями, удобную работу с функциональными методами и лаконичный синтаксис, что делает разработку более быстрой и менее подверженной ошибкам.
* **Альтернативы**: **Java** является еще одним популярным языком для Android-разработки, однако Kotlin предлагает более простое и выразительное решение для современных приложений. [4]

## 2.2. Библиотеки и фреймворки

**PyTorch** — это популярная библиотека для работы с нейронными сетями, которая позволяет легко строить, обучать и развертывать модели машинного обучения. Она поддерживает динамическое вычисление графов, что делает её удобной для быстрого прототипирования.

* **Почему выбран**: PyTorch был выбран за свою гибкость, удобство в обучении нейронных сетей и активную поддержку мобильных платформ через **PyTorch Mobile**. PyTorch также имеет развитую документацию и сообщество, что значительно облегчает процесс разработки и внедрения.
* **Альтернативы**: **TensorFlow** — конкурент PyTorch, предоставляющий аналогичные возможности, однако для мобильных приложений PyTorch Mobile обеспечивает более простую интеграцию и поддержку. [5]

**TorchScript** — это способ сериализации моделей PyTorch, который позволяет использовать их на мобильных устройствах. С помощью TorchScript модель PyTorch конвертируется в формат, который можно легко загрузить и выполнить на Android.

* **Почему выбран**: TorchScript выбран для конвертации модели в формат, совместимый с Android-приложением. Это позволяет использовать ту же модель, которая была обучена в Python, в мобильном приложении, обеспечивая высокий уровень производительности и совместимости.
* **Альтернативы**: **ONNX** (Open Neural Network Exchange) также является форматом для сериализации моделей и может быть использован для работы с TensorFlow, PyTorch и другими библиотеками. Однако для использования PyTorch в Android приложении TorchScript является более естественным выбором.

## 2.3. Инструменты мобильной разработки

**Android Studio** — это официальная среда разработки для Android, которая предоставляет весь необходимый инструментарий для создания приложений, включая редактор кода, эмуляторы, инструменты для отладки и анализа производительности.

* **Почему выбран**: Android Studio это официальная среда разработки для Android, которая поддерживает все необходимые инструменты для интеграции с PyTorch и OpenCV.
* **Альтернативы**: **IntelliJ IDEA** также поддерживает Android-разработку, но Android Studio предоставляет более тесную интеграцию с платформой Android.

**Gradle** — это инструмент для автоматизации сборки, который используется для управления зависимостями, настройки и сборки проекта. Он широко используется в экосистеме Android.

* **Почему выбран**: Gradle это стандартный инструмент для управления зависимостями и сборки проектов на Android, и он идеально подходит для интеграции библиотек, таких как OpenCV, PyTorch и CameraX.
* **Альтернативы**: **Maven** — еще один инструмент для управления зависимостями, однако Gradle предоставляет больше возможностей для гибкой настройки сборки в Android-приложениях.

**CameraX** — это инструменты для работы с камерами на Android. CameraX упрощает работу с камерой, обеспечивая простоту использования и поддержку различных типов устройств.

* **Почему выбран**: CameraX был выбран для работы с камерой, так как он упрощает захват изображений, что особенно важно для приложения, использующего компьютерное зрение для распознавания номеров автобусов. [6]
* **Альтернативы**: **Camera2 API** предоставляет более низкоуровневый доступ к камере, что дает больше возможностей, но требует более сложной настройки. CameraX является более высокоуровневым API, которое подходит для большинства случаев использования.

**OpenCV (for android)** — это библиотека для обработки изображений и видео в реальном времени. В контексте мобильных приложений, она предоставляет удобный набор инструментов для работы с изображениями и видео.

* **Почему выбран**: OpenCV был выбран для обработки изображений с камеры, так как эта библиотека предоставляет широкий функционал для манипуляции изображениями, а также для предварительной обработки данных перед подачей их в нейронную сеть.
* **Альтернативы**: **TensorFlow Lite** также предоставляет инструменты для обработки изображений, но OpenCV предлагает более широкие возможности и гибкость в области обработки изображений.

## 2.4. Инструменты тестирования

**Android Emulator** — это инструмент для тестирования Android-приложений на различных устройствах без необходимости использовать физическое устройство.

* **Почему выбран**: Android Emulator был выбран для тестирования приложения, так как он позволяет имитировать работу различных Android-устройств, что важно для проверки совместимости и функциональности приложения.
* **Альтернативы**: ф**изическое устройство** — может быть использовано для тестирования, однако Android Emulator предоставляет более быстрый и удобный способ тестирования на разных устройствах.

## 2.5. Инструмент разметки датасета

**Computer Vision Annotation Tool (CVAT)** — это веб-инструмент с открытым исходным кодом, предназначенный для разметки изображений и видео в задачах компьютерного зрения. CVAT поддерживает такие типы аннотаций, как прямоугольные рамки (bounding boxes), многоугольники, полилинии, ключевые точки и трассировка объектов во времени.

* **Почему выбран**: CVAT предоставляет удобный и интуитивно понятный интерфейс для аннотирования изображений, позволяет создавать задачи, распределять их между участниками, поддерживает экспорт в различные форматы (COCO, Pascal VOC, CVAT for images), а также имеет встроенную валидацию и отслеживание качества разметки.
* **Альтернативы**: LabelImg — лёгкий инструмент для создания bounding box аннотаций, однако он менее удобен при работе с большими датасетами. Label Studio — более универсален и настраиваем, но требует дополнительной конфигурации и не так хорошо оптимизирован под задачи компьютерного зрения.

# ****3. Описание предметной области****

## 3.1. Компьютерное зрение

Компьютерное зрение — это область искусственного интеллекта, направленная на создание алгоритмов и систем, которые могут интерпретировать и анализировать содержимое цифровых изображений и видео. Основная задача состоит в том, чтобы компьютер «понимал» изображения или видеопотоки так же, как это делает человек. С точки зрения техники, компьютерное зрение включает в себя обработку данных (цифровых изображений) и применение алгоритмов для выполнения задач, таких как распознавание, сегментация, отслеживание и анализ объектов.

Ключевая особенность компьютерного зрения — способность машин анализировать визуальные данные в реальном времени и адаптироваться к динамически изменяющимся условиям. Это качество имеет решающее значение для задач, связанных с городской средой, где изображения часто содержат большое количество объектов и сложные условия освещения.

### 3.1.1. Задача распознавания объектов

Распознавание объектов (Object Detection) — одна из центральных задач в компьютерном зрении. Её цель состоит в определении местоположения объектов определённых классов на изображении. Задача включает два этапа: детекция объектов и классификация объектов. [7]

На рисунке 1 представлена схема работы одного из алгоритмов распознавания объектов – R-CNN:

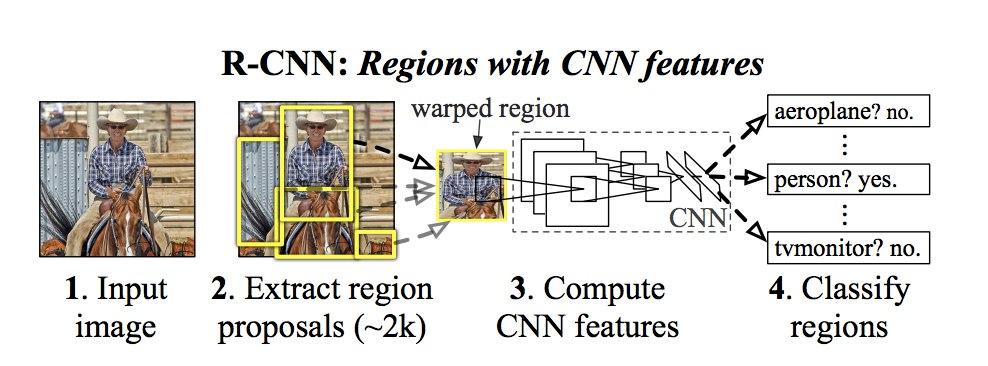


Рисунок 1 — Схема работы алгоритма R-CNN

**Терминология:**

* **Ограничивающая рамка (bounding box)** – координаты, ограничивающие определенную область изображения, – чаще всего в форме прямоугольника. Может быть представлена в виде четырёх координат в двух форматах: центрированный () и обычный ().
* **Гипотеза (Proposal), P** – определенный регион изображения (заданный с помощью ограничивающей рамки), в котором предположительно находится объект.
* **IoU (Intersection-over-Union)** – метрика степени пересечения между двумя гипотезами. Определяется как отношение области пересечения (area of overlap) и области объединения (area of union) их рамок. На рисунке 2 приведена наглядная иллюстрация определения метрики IoU.



Рисунок 2 — Определение метрики IoU

**Детекция объектов** — это процесс нахождения и выделения области, в которой расположен объект интереса, на изображении или видео. Она включает не только идентификацию наличия объекта, но и определение его местоположения. Основные этапы детекции:

1. Локализация объекта**:** определение координат ограничивающей рамки, в которой предположительно находится объект.
2. Анализ контекста**:** использование дополнительных признаков (большего количества гипотез) для подтверждения наличия объекта.

Для обнаружения объекта изображение делится на фрагменты в виде прямоугольных рамок, что позволяет модели выявлять ключевые признаки и определять расположение объекта. Эффективность модели оценивается с помощью метрики IoU, которая измеряет совпадение между истинной рамкой объекта и предполагаемой моделью. Этот метод оценки иллюстрируется на рисунке 3.

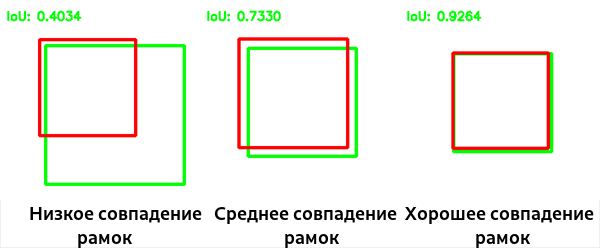


Рисунок 3 — Степень совпадения рамок

**Классификация объектов** — это процесс присвоения объекта к одному из предопределённых классов. Основные этапы детекции **классификации:**

1. **Подготовка данных:**
   * Удаление шумов и улучшение качества изображения (например, выравнивание освещения).
   * Преобразование выделенной области в формат, пригодный для подачи в нейронную сеть.
2. **Извлечение признаков:**
   * Использование алгоритмов глубокого обучения, чтобы извлечь особенности объекта, такие как контуры и текстуры.
3. **Присвоение класса:**
   * На основе извлечённых признаков объект классифицируется как определённый номер или символ.

В задаче распознавания номеров автобусов детекция помогает выделить область табло автобуса, после чего происходит дальнейший анализ для считывания номера. Это особенно важно в условиях городской среды, где объекты могут частично перекрываться или находиться в движении.

### 3.1.2. Задача распознавания текста

Распознавание текста (Optical Character Recognition, OCR) направлено на извлечение текстовой информации из изображений. Основная сложность OCR состоит в том, что текст может быть написан разными шрифтами, находиться под углом или быть частично затенённым. [8]

На рисунке 4 представлена общая схема работы алгоритмов распознавания текста:

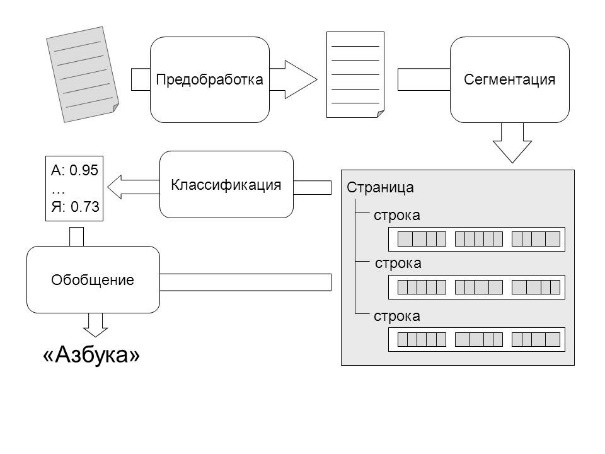


Рисунок 4 — Схема работы алгоритма, решающий задачи распознавания текста

1. Предобработка изображения:
   * Градации серого: преобразование изображения в оттенки серого для уменьшения размерности данных.
   * Бинаризация: преобразование изображения в двоичный формат (черное-белое) для акцентирования текста.
   * Удаление шумов: фильтрация мелких дефектов, вызванных внешними условиями (например, дождем или пылью).
   * Выравнивание: коррекция перспективных искажений текста из-за углов съемки.
2. Детекция текста:
   * На данном этапе выделяются области, содержащие текст
3. Распознавание текста (OCR):
   * После выделения текстовой области используется Optical Character Recognition - технология преобразования текстовой информации в цифровой, машиночитаемый текст.
4. Постобработка результатов
   * Исправление ошибок: использование словарей или статистических моделей для устранения ошибок распознавания.
   * Устранение ложных срабатываний: фильтрация неверных областей, например, случайных шумов, распознанных как текст.

В контексте задачи распознавания номеров автобусов OCR используется для извлечения числовой информации с табло автобуса, что требует адаптации алгоритмов к условиям городской среды, например, к низкому качеству изображения или яркому солнечному свету.

## 3.2. Свёрточные нейронные сети

Свёрточные нейронные сети (Convolutional Neural Networks, CNN) — это класс глубоких нейронных сетей, которые особенно эффективны в задачах обработки изображений. CNN автоматически извлекают признаки из изображений, такие как контуры, текстуры и формы, что делает их идеальными для задач компьютерного зрения. [9]

CNN состоят из нескольких основных компонентов:

* **Свертка** — основной строительный блок CNN. Она заключается в применении фильтров к входным данным для выделения различных признаков, таких как края, текстуры и формы. Фильтр скользит по изображению и вычисляет свертку, создавая карту признаков. Этот процесс позволяет выделить важные детали изображения, которые затем используются для дальнейшей обработки. С математической точки зрения результатом операции свёртки над матрицами *A* (размером ) и *B* (размером ) является матрица *C* (размером ), элементы который вычисляются следующей формулой: . Работа операции свертки для двух матриц представлена на рисунке 5.

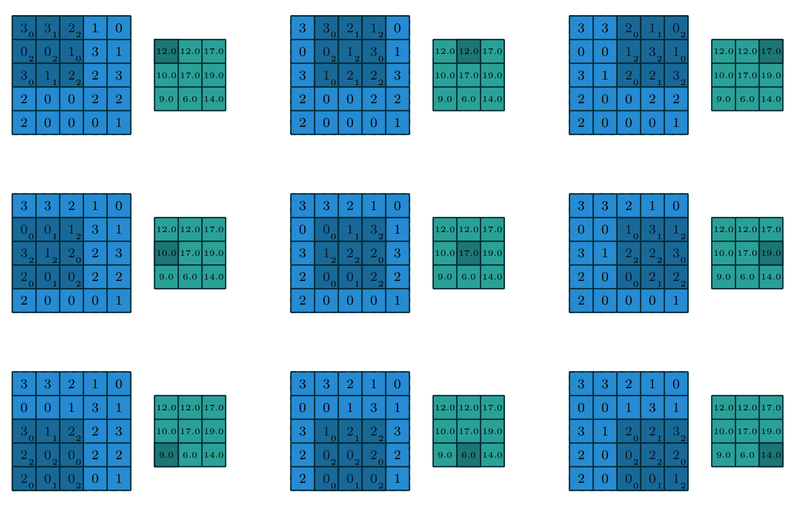


Рисунок 5 — Пример свертки двух матриц

* После свертки применяется **функция активации**, которая добавляет нелинейность в модель. Это помогает ей лучше справляться с сложными задачами, так как нелинейные функции позволяют нейронной сети моделировать более сложные зависимости.
* **Пулинг** (субдискретизация) уменьшает размер карты признаков, сохраняя наиболее значимые признаки. Это помогает уменьшить количество параметров и вычислительных затрат, сохраняя при этом важные характеристики изображения. На рисунке 6 представлена работа операции пулинга с функцией максимума.



Рисунок 6 — Пример операции пулинга с функцией максимума

Архитектуры CNN предоставляют гибкость в настройке модели для работы на устройствах с ограниченными вычислительными ресурсами, например, на смартфонах. В задаче распознавания номеров автобусов CNN используются для детекции табло и распознавания текста. [10]

## 3.3. Архитектура свёрточной нейронной сети

В общем виде CNN модели представляют собой входной слой и выходной слой, а также расположенный между ними набор слоёв свёртки, слоёв активации, слоёв пулинга и полносвязных слоёв. Общую схему работы CNN моделей с классической архитектурой можно увидеть на рисунке 7.

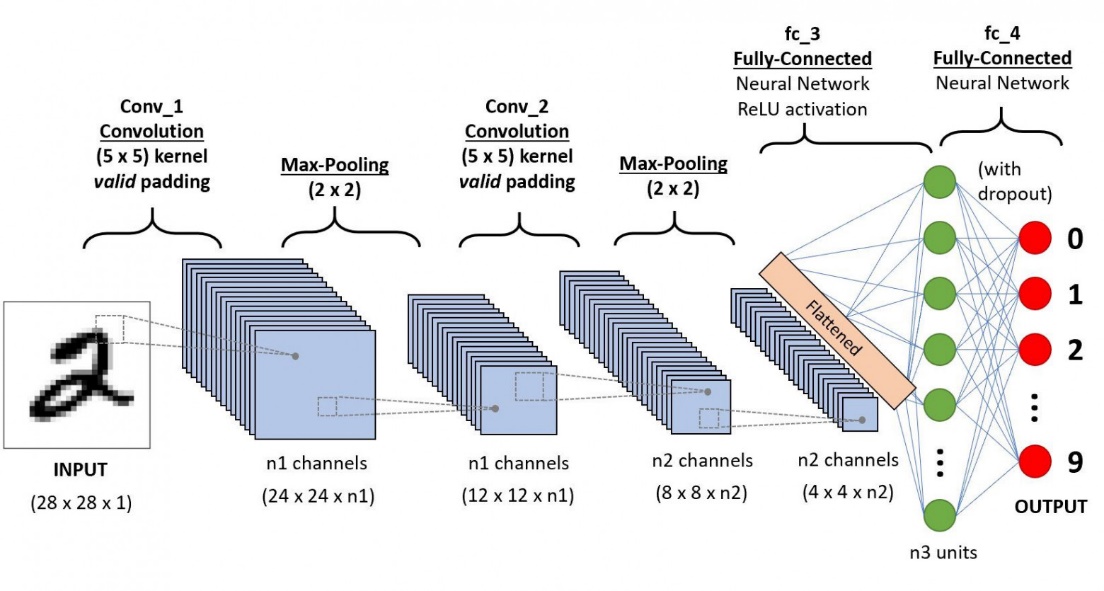


Рисунок 7 — Схема работы CNN моделей

Перед подачей в сеть изображения проходят предобработку: изменение размера (например, до 128x128 пикселей), нормализация значений пикселей и преобразование формата данных в тензор. **Входной слой** принимает данный тензор (то есть массив чисел) для дальнейшей обработки нейронной сетью.

**Сверточные слои** выполняют свертку с различными фильтрами для выделения признаков. Каждый сверточный слой создает несколько карт признаков, которые затем передаются на следующий слой. Эти слои являются основными компонентами CNN и выполняют основную работу по выделению признаков из изображения. Каждый сверточный слой может использовать несколько фильтров, каждый из которых выделяет разные признаки. Для распознавания текста на автобусах свёрточные слои помогают выделять ключевые элементы, такие как формы цифр. Глубокие слои могут изучать более сложные абстракции, например, стиль шрифта или расположение текста.

**Слои активации** применяют функцию активации (например, ReLU) к картам признаков, добавляя нелинейность в модель. Это позволяет модели лучше справляться с сложными задачами, так как нелинейные функции позволяют нейронной сети моделировать более сложные зависимости.

**Пулинговые слои** уменьшают размер карт признаков, сохраняя важные признаки и уменьшая вычислительные затраты. Это помогает модели быть более эффективной и быстрой, особенно при обработке больших изображений.

**Полносвязные слои** соединяют все нейроны предыдущего слоя с каждым нейроном текущего слоя. Они используются для окончательной классификации или регрессии. Эти слои являются завершающими компонентами CNN и выполняют основную работу по принятию решений на основе выделенных признаков. В задаче распознавания номера автобуса выход полносвязных слоев может быть интерпретирован как вероятности принадлежности изображения к различным номерам.

**Выходной слой** генерирует окончательный результат, для задачи распознавания текста это может быть один из заранее заданных классов (например, цифры от 0 до 9 и буквы).

## 3.4. Обучение свёрточной нейронной сети

**Датасет**: для обучения модели необходимо собрать и разметить набор данных. В данному случае это изображения автобусов с чётко видимыми номерами. Данные проходят предобработку: кадрирование для выделения области с номером, изменение размера, нормализация. Для улучшения качества обучения применяется аугментация данных — добавление шумов, повороты, изменение яркости и контраста, чтобы сеть могла адаптироваться к реальным условиям.

**Гиперпараметры сети**:

* **Размер свёрток** определяет область изображения, которую анализирует фильтр за один проход. В выбранной архитектуре используется размер 3x3. Это значение позволяет эффективно выделять локальные признаки, такие как линии, границы и мелкие текстовые элементы, не увеличивая избыточность параметров.
* **Количество фильтров** в каждом слое определяет, сколько различных признаков будет извлечено на каждом уровне. В начальных слоях используется 32 фильтра для захвата базовых признаков (например, текстур и краёв). В средних слоях это число увеличивается до 64, чтобы извлечь более сложные паттерны. В глубоких слоях 128 фильтров используются для формирования высокоуровневых признаков, таких как формы букв или цифр.
* **Шаг свёртки** определяет, насколько фильтр смещается по изображению при обработке. Выбранный шаг равен 1, что позволяет сохранить высокую пространственную детализацию изображения.
* **Размер batch** определяет количество примеров, обрабатываемых моделью за один проход. В данном случае выбрано значение 32, которое позволяет эффективно использовать ресурсы графического процессора, обеспечивая стабильность вычислений и плавную сходимость модели.
* ReLU (Rectified Linear Unit) используется как **функция активации**. Её простота и эффективность обусловлены тем, что она устраняет проблему затухания градиента и ускоряет обучение модели. ReLU позволяет сохранять значимую информацию для последующих слоёв, оставляя только положительные значения.
* Алгоритм Adam (Adaptive Moment Estimation) выбран в качестве **оптимизатора**. Он автоматически корректирует шаг градиента для каждого параметра, что делает его подходящим для задач с шумными данными.
* **Функция потерь:** Cross-Entropy Loss используется для измерения расхождения между предсказанными вероятностями и истинными метками. Эта функция оптимальна для задач классификации, таких как определение номера автобуса. Она минимизирует ошибку, способствуя повышению точности модели.
* **Пуллинг:** MaxPooling (размер 2x2, шаг 2) используется для уменьшения размерности карты признаков, сохраняя наиболее важные из них. Этот подход снижает вычислительные затраты и уменьшает риск переобучения, сохраняя при этом ключевые характеристики текста.

Процесс обучения заключается в подаче размеченных изображений на вход сети, вычислении функции потерь, обновлении весов с помощью градиентного спуска и проверки результатов на валидационной выборке. Обучение продолжается до достижения оптимального значения точности. [11, 12]

## 3.6. Работа сети после завершения обучения

После обучения модель экспортируется в формат, подходящий для мобильного приложения (TorchScript). Это позволяет запускать модель на устройстве без серверной обработки. Интеграция осуществляется с использованием PyTorch Mobile.

Изображения с камеры смартфона поступают в приложение, где проходят предобработку, аналогичную той, что использовалась при обучении. Это обеспечивает соответствие формата данных и корректность работы сети.

Сеть предсказывает номер автобуса на изображении, преобразует результат в текстовую или голосовую форму. Для удобства пользователей, особенно слабовидящих, информация отображается крупным шрифтом и озвучивается через аудиоинтерфейс.

# 4. Практическая часть

## 4.1. Подготовка датасета

Для реализации задачи распознавания типа общественного транспорта и его маршрутного номера был сформирован собственный датасет. Изображения получались в реальных условиях городской среды с использованием камер мобильных телефонов.

В сборе данных принимали участие несколько человек, что позволило расширить вариативность условий съёмки. Из-за использования различных моделей смартфонов в датасет попали фотографии с разными фокусными расстояниями, разрешением и цветопередачей. Это обеспечило дополнительную устойчивость нейросети к реальным условиям эксплуатации.

Съёмка проводилась в разное время суток и при различных погодных условиях, что способствовало формированию датасета, отражающего реальные сценарии использования.

Целевым ориентиром при сборе данных было достижение объёма не менее 500 качественных изображений с чётко различимыми объектами и маршрутными номерами. Изображения с плохим качеством (размытость, засветка, неполное попадание транспорта в кадр) были удалены из датасета.

**Разметка и предобработка датасета:**

В рамках данной работы использовался формат **CVAT for images 1.1**, который представляет собой набор XML-файлов и изображений, сопровождаемых информацией о разметке. [13]

Каждое изображение размечалось вручную с применением прямоугольных ограничивающих рамок (**bounding boxes**) по двум категориям:

* **Тип транспорта**: каждому транспортному средству на изображении соответствовала рамка с категорией из следующих четырёх классов: **автобус**, **троллейбус**, **трамвай**, **маршрутка** (а также “ **unknown**). Это заготовка под потенциальный классификатор.
* **Маршрутный номер**: отдельной рамкой выделялась область, где находится маршрутный номер (обычно на передней части транспорта, в виде таблички или светодиодного дисплея). Эта аннотация использовалась на этапе OCR для извлечения текста. Пример разметки можно увидеть на рисунке 8.

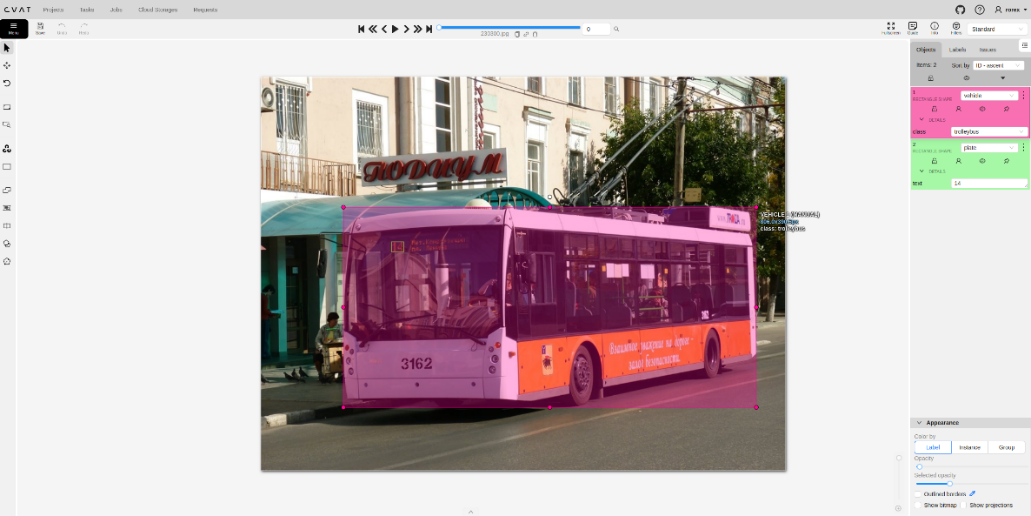


Рисунок 8 — Пример разметки данных в CVAT

CVAT позволяет организовать удобный процесс разметки: изображения подгружаются в проект, каждому объекту на изображении присваивается класс, а все аннотации экспортируются в нужном формате. CVAT for images 1.1 сохраняет аннотации в виде XML-файлов, по одному на каждое изображение. В каждом таком файле содержится информация о позициях объектов, их классах и других атрибутах:

<annotations>

<version>1.1</version>

<meta>

<task>

<labels>

<label>

<name>bus</name>

</label>

<label>

<name>route\_number</name>

</label>

</labels>

</task>

</meta>

<image id="0" name="IMG\_001.jpg" width="1920" height="1080">

<box label="bus" xtl="230.5" ytl="112.7" xbr="870.1" ybr="654.3" />

<box label="route\_number" xtl="310.2" ytl="140.6" xbr="410.8" ybr="180.2" />

</image>

</annotations>

Однако данный формат не является напрямую пригодным для использования в модели, поэтому была реализована процедура конвертации аннотаций в упрощённый формат CSV-подобного файла через небольшой python-скрипт. Он извлекает из разметки координаты маршрутного номера (в нормализованном виде), категорию транспортного средства и текст самого номера, если он указан. Аннотации преобразуются в строки вида: **<имя файла>,x>,<y>,<w>,<h>,<маршрутный номер>,**  где x, y, w, h — нормализованные координаты центра и размеры рамки с номером. Это решение позволило упростить последующую загрузку и использование аннотированных данных в модели.

## 4.2. Архитектура нейронной сети

Модель представляет собой многозадачную свёрточную нейронную сеть (Multi-Task Learning CNN), одновременно решающую три задачи:

1. Классификация типа транспортного средства;
2. Детекция bounding box (детекция номерного знака);
3. Распознавание текста (OCR) на транспортном средстве.

Архитектура использует общий энкодер (backbone) для извлечения признаков и специализированные головы (heads) для каждой из задач, что позволяет эффективно использовать вычислительные ресурсы и улучшать обобщающую способность модели за счёт совместного обучения. [12, 14]

### 4.2.1. Общий извлекатель признаков

Основу модели составляет сверточный блок, состоящий из последовательности из четырёх идентичных модулей. Каждый модуль реализует базовую последовательность операций: двумерная свёртка, пакетная нормализация, нелинейная активация ReLU и операция субдискретизации (MaxPooling). Такая конструкция позволяет поэтапно извлекать и уплотнять пространственные признаки изображения, уменьшая разрешение входного изображения с 512×512 до 32×32 при увеличении количества каналов с 3 до 512.

class ConvBlock(nn.Module):

def \_\_init\_\_(self, in\_channels, out\_channels, kernel\_size=3, stride=1, padding=1):

super().\_\_init\_\_()

self.block = nn.Sequential(

nn.Conv2d(in\_channels, out\_channels, kernel\_size, stride, padding),

         nn.BatchNorm2d(out\_channels),

         nn.ReLU(inplace=True),

         nn.MaxPool2d(2))

def forward(self, x):

      return self.block(x)

class MultiTaskModel(nn.Module):

def \_\_init\_\_(self,num\_classes=5,ocr\_vocab\_size=22+1, freeze\_cls=True):

super().\_\_init\_\_()

self.num\_classes = num\_classes

self.ocr\_vocab\_size = ocr\_vocab\_size

self.freeze\_cls = freeze\_cls

self.backbone = nn.Sequential(

ConvBlock(3, 64),

ConvBlock(64, 128),

ConvBlock(128, 256),

ConvBlock(256, 512))

### 4.2.2. Классификатор типа транспорта

Модуль классификации определяет тип транспортного средства (автобус, трамвай, троллейбус и т.д.) перед распознаванием номера маршрута. Из-за ограниченности датасета данный функционал реализован лишь для одного класса (троллейбусы). Однако, потенциальными преимуществами такого подхода являются:

1. **Фильтрация объектов** - исключает обработку посторонних объектов (легковые авто, грузовики);
2. **Контекст для OCR** - знание типа транспорта помогает предсказывать возможные номера маршрутов и необходимо в случае увеличения функциональности приложения (если потребуется разрешать конфликты, когда разные типы транспорта имеют одинаковый номер маршрута).

Для определения типа транспортного средства используется адаптивный глобальный пуллинг, за которым следует полносвязный слой. Глобальный пуллинг позволяет агрегировать пространственные признаки в один вектор, обеспечивая инвариантность к положению объекта на изображении. Финальный линейный слой выдаёт вектор логитов по числу классов, соответствующий вероятностному распределению по типам транспорта.

self.cls\_head = nn.Sequential(

nn.AdaptiveAvgPool2d((1, 1)),

nn.Flatten(),

nn.Linear(512, num\_classes))

### 4.2.3. Детекция bounding box

Модуль детекции bounding box определяет точное местоположение номера маршрута на изображении транспортного средства.

Модель точно находит область с номером маршрута среди других элементов, что исключает обработку постороннего текста (реклама, надписи на кузове) и позволяет сосредоточить ресурсы OCR на нужной области.

При таком подходе нейросеть обучается автоматически определять оптимальные зоны поиска номера в зависимости от типа транспорта, учитывая характерные пространственные закономерности: типичное расположение номерных знаков относительно геометрии транспортного средства, преобладающую ориентацию (вертикальную или горизонтальную), а также устойчивые пространственные соотношения между областью номера и ключевыми элементами конструкции.

Задача локализации номерного знака решается с помощью сверточного слоя и полносвязной регрессионной головы. После свёртки пространственных признаков применяется оператора выпрямления (flattening), после чего следует полносвязный слой, выдающий четыре нормализованных координаты: центр и размеры предполагаемого прямоугольника. Использование сигмоиды обеспечивает сжатие предсказанных координат в диапазон [0, 1].

self.bbox\_head = nn.Sequential(

nn.Conv2d(512, 64, 3, padding=1),

nn.Flatten(),

nn.Linear(64 \* 32 \* 32, 4),

nn.Sigmoid())

### 4.2.4. Модуль распознавания текста

Модуль OCR отвечает за распознавание номера маршрута на общественном транспорте. Он анализирует область изображения, выделенную bounding box, и преобразует визуальную информацию в текстовую строку.

Для извлечения текстовой информации из изображения используется специализированная архитектура с элементами последовательной обработки:

* Признаки изображения усредняются по вертикали, сохраняя горизонтальную структуру (ширину), что позволяет представить данные как последовательность.
* Преобразованные признаки подаются на однонаправленную LSTM-ячейку, формирующую контекстуализированные векторы признаков по временной оси.
* Далее применяется линейная проекция и механизм внимания, позволяющий модели фокусироваться на наиболее релевантных символах последовательности.
* Финальный полносвязный слой проецирует признаки в пространство словаря символов (буквы и цифры), формируя логиты для каждой позиции.

self.ocr\_feat\_proj = nn.Linear(512, 64)

self.ocr\_lstm = nn.LSTM(64, 32, batch\_first=True, bidirectional=False)

self.ocr\_attention = nn.Sequential(

nn.Linear(32, 1),

nn.Softmax(dim=1))

self.ocr\_fc = nn.Linear(32, ocr\_vocab\_size)

### 4.2.5. Прямой проход

Прямой проход модели определяет, как входное изображение преобразуется в выходные данные каждой из трёх задач: классификации, локализации и распознавания текста.

На вход модель принимает тензор изображения размерности [B, 3, H, W], где B — размер батча. Сначала изображение проходит через сверточную нейросеть (backbone), затем обрабатывается классификатором и детектором рамки:

features = self.backbone(x)

cls\_out = self.cls\_head(features)

bbox\_out = self.bbox\_head(features)

**Для OCR используется цепочка модулей:**

ocr\_feat = F.adaptiveavg\_pool2d(features, (1, features.size(3)))

ocr\_feat = ocr\_feat.squeeze(2).permute(0, 2, 1)

ocr\_feat = self.ocr\_featproj(ocr\_feat)

ocr\_feat, \_ = self.ocr\_lstm(ocrfeat)

ocr\_out = self.ocr\_fc(ocrfeat)

Выход ocr\_out представляет логиты для каждого символа в каждой позиции. Он обрабатывается CTC-декодером при инференсе и используется для расчёта CTC-потерь при обучении. Выход cls\_out — логиты размерности [B, num\_classes], которые позже обрабатываются через softmax. Выход bbox\_out представляет собой вектор [x\_center, y\_center, width, height], нормализованный в диапазоне [0, 1].

### 4.2.6. Функции потерь и постобработка

Для обучения многозадачной модели применяется совокупная функция потерь, включающая три слагаемых: классификационную, регрессионную и CTC-потерю. Все они направлены на одновременное обучение разных голов модели.

loss\_cls = F.cross\_entropy(cls\_out, cls\_target)

loss\_bbox = F.mse\_loss(bbox\_out, bbox\_target) \* 10

loss\_ocr = self.ocr\_ctc\_loss(ocr\_out, ocr\_target, ocr\_lengths)

return loss\_cls + loss\_bbox + loss\_ocr, loss\_cls, loss\_bbox, loss\_ocr

После завершения обучения модель применяется для реального распознавания транспортных средств на изображениях. Метод predict реализует инференс с постобработкой результатов. На первом этапе выполняется прямой проход через модель (forward), в ходе которого получаются логиты для каждой из трёх задач: классификации, предсказания координат рамки и OCR.

Для определения класса транспортного средства логиты преобразуются в вероятности с помощью softmax, после чего выбирается наиболее вероятный класс. Однако, если уверенность модели (prob[class]) ниже заданного порога (cls\_threshold, по умолчанию 0.6), объект маркируется как «неизвестный» путём принудительного присваивания специального индекса класса.

cls\_probs = F.softmax(cls\_out, dim=1)

cls\_pred = torch.argmax(cls\_probs, dim=1)

cls\_pred[cls\_probs[torch.arange(len(cls\_probs)), cls\_pred] < cls\_threshold] = 4

Bounding box принимается только в случае, если его площадь превышает порог (например, 1% от общей площади изображения), что исключает заведомо шумные или вырожденные предсказания.

plate\_bbox = bbox\_out if (bbox\_out[:, 2] \* bbox\_out[:, 3] > 0.01).all() else None

OCR-логиты также проходят через softmax, после чего для каждой временной позиции выбирается наиболее вероятный символ. В дополнение к этому рассчитывается максимальная уверенность (confidence) по всей последовательности. Если она ниже заданного порога (ocr\_confidence\_threshold, по умолчанию 0.4), текст считается нераспознанным. В противном случае предсказанная последовательность индексов декодируется в строку с помощью функции decode\_plate.

ocr\_probs = F.softmax(ocr\_out, dim=2)

top\_char = torch.argmax(ocr\_probs, dim=2)

top\_conf = torch.max(ocr\_probs, dim=2).values

if top\_conf.item() < ocr\_confidence\_threshold:

plate\_text = "Не распознано"

else:

platetext = self.decodeplate(top\_char[0])

return cls\_pred, plate\_bbox, plate\_text

Метод decode\_plate преобразует предсказанные индексы в строку, исключая спецсимволы (например, "blank" из CTC), и использует заранее определённый алфавит, включающий цифры и буквы кириллицы, допустимые в номерах транспортных средств: 0123456789АВЕКМНОРСТУХ.

def decode\_plate(self, char\_indices):

chars = "0123456789АВЕКМНОРСТУХ"  # Словарь символов

return ''.join([chars[i] for i in char\_indices if i < len(chars)])

## 4.3. Обучение многозадачной модели

Для решения задачи, объединяющей классификацию транспорта, детекцию номерного знака и его распознавание, была реализована специализированная процедура обучения модели с поддержкой mixed precision, логированием потерь и сохранением лучших результатов. [14]

Процесс начинается с загрузки конфигурации из YAML-файла, содержащего параметры модели, обучения и словарь допустимых символов для OCR. Далее инициализируется многозадачная модель MultiTaskModel.

with open("config.yaml") as f:

config = yaml.safe\_load(f)

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

model = MultiTaskModel(

num\_classes=5,

ocr\_vocab\_size=len(config['model']['vocab']) + 1).to(device)

Для подготовки данных используется кастомный датасет VehicleDataset,который:

* загружает изображения и метки из текстового файла;
* нормализует координаты bounding box в относительные значения;
* кодирует текстовые метки с учётом CTC loss (blank-символ и паддинг);
* поддерживает работу с последовательностями переменной длины через collate\_fn.

train\_dataset = VehicleDataset(

root\_dir="data/train",

vocab=config['model']['vocab'],

image\_size=config['image']['size'])

Обучение производится с помощью оптимизатора Adam и планировщика ReduceLROnPlateau, который адаптирует скорость обучения при отсутствии улучшения потерь. Все вычисления осуществляются на GPU (при наличии), с использованием torch.amp для автоматического выбора точности вычислений. Это снижает требования к видеопамяти и ускоряет обучение без потери качества.

optimizer = Adam(model.parameters(), lr=config['training']['lr'])

scheduler = lr\_scheduler.ReduceLROnPlateau(

optimizer,

mode='min',

factor=0.5,

patience=3)

scaler = torch.amp.GradScaler(device='cuda' if torch.cuda.is\_available() else 'cpu')

На каждой эпохе модель переводится в режим обучения, затем выполняется проход по батчам с подсчётом потерь отдельно для каждой задачи: классификации (CrossEntropyLoss), регрессии (MSELoss) и OCR (CTCLoss). Итоговая функция потерь представляет собой взвешенную сумму этих трёх составляющих. Для повышения стабильности и предотвращения градиентного взрыва применяется обрезка градиентов (clip\_grad\_norm\_).

for epoch in range(config['training']['epochs']):

model.train()

epoch\_loss = 0.0

for batch in progress\_bar:

images, cls\_targets, bbox\_targets, ocr\_targets, ocr\_lengths = [x.to(device, non\_blocking=True) for x in batch]

optimizer.zero\_grad(set\_to\_none=True)

with torch.amp.autocast(device\_type=device.type):

cls\_out, bbox\_out, ocr\_out = model(images)

loss, loss\_cls, loss\_bbox, loss\_ocr = model.compute\_losses(

cls\_out,  # Выход классификатора

bbox\_out,  # Выход детектора bbox

ocr\_out,  # Выход OCR

cls\_targets.view(-1),  # Целевые классы

bbox\_targets,  # Целевые bbox

ocr\_targets,  # Целевые тексты

ocr\_lengths  # Длины текстов)

…

После каждой эпохи рассчитывается средняя потеря, и если она оказывается ниже предыдущей, текущая модель сохраняется как лучшая. Это позволяет впоследствии использовать наиболее удачные веса, не переобучая модель.

avg\_loss = epoch\_loss / len(train\_loader)

scheduler.step(avg\_loss)

if avg\_loss < best\_loss:

best\_loss = avg\_loss

torch.save({

'epoch': epoch,

'model\_state\_dict': model.state\_dict(),

'optimizer\_state\_dict': optimizer.state\_dict(),

'loss': best\_loss,

}, "best\_model.pth")

## 4.4. Экспорт модели

Из YAML-файла считывается информация о словаре допустимых символов для задачи распознавания текста. Размер словаря нужен для корректной инициализации модели, так как последняя линейная проекция OCR-головы зависит от количества выходных классов (ocr\_vocab\_size).

Создаётся экземпляр модели MultiTaskModel с параметрами, соответствующими конфигурации. Затем из файла best\_model.pth загружается сохранённое состояние обученной модели.

Модель переводится в режим eval, что отключает слои регуляризации (Dropout, BatchNorm и др.), чтобы обеспечить стабильное поведение при инференсе. [14]

with open("config.yaml") as f:

config = yaml.safe\_load(f)

model = MultiTaskModel(

num\_classes=5,

ocr\_vocab\_size=len(config['model']['vocab']) + 1)

checkpoint = torch.load("best\_model.pth", map\_location="cpu")

model.load\_state\_dict(checkpoint['model\_state\_dict'])

model.eval()

TorchScript предлагает два способа преобразования модели: tracing и scripting. В данном случае используется torch.jit.script, так как модель содержит условные конструкции (например, при возврате разных выходов), которые не обрабатываются корректно в режиме tracing. Преимущество script() заключается в том, что он анализирует сам код модели (включая ветвления, циклы и условия), превращая его в статическое представление, пригодное для выполнения вне среды Python.

Полученный ScriptModule сериализуется и сохраняется в файл model.pt. Этот файл может быть затем загружен и выполнен в любом окружении, поддерживающем libtorch или torch.jit.

scripted\_model = torch.jit.script(model)

scripted\_model.save("model.pt")

print("TorchScript-модель успешно сохранена")

## 4.5. Реализация android-приложения

### 4.5.1. Проектирование пользовательского интерфейса

Для реализации пользовательского интерфейса приложения был использован ConstraintLayout, обеспечивающий гибкое позиционирование элементов на экране и адаптивность под различные размеры устройств. [15]

Основным визуальным компонентом интерфейса является элемент PreviewView из библиотеки androidx.camera.view, который занимает верхнюю часть экрана и служит для отображения изображения с камеры в реальном времени. Размер PreviewView ограничен снизу расположением текстового поля с результатами распознавания, что позволяет визуально отделить вывод данных от видеоизображения.

Под областью с камерой расположены два текстовых элемента TextView:

* **tvResult** — отображает результаты распознавания (например, тип транспорта и номерной знак). Для повышения читаемости текста на фоне динамичного видео ему задан полупрозрачный чёрный фон (#AA000000), белый цвет текста, размер 24sp и жирное начертание. Текст расположен непосредственно под видеоизображением, что предотвращает слияние с изменчивым фоном камеры и обеспечивает удобное восприятие информации пользователем.
* **tvStatus** — служит для вывода текущего состояния работы приложения. Аналогично tvResult, тексту присвоен полупрозрачный фон и белый цвет, но размер текста меньше — 18sp. Элемент находится в нижней части экрана.

### 4.5.2. Программная реализация

При запуске приложения происходит загрузка необходимых компонентов, включая нейросетевую модель в формате TorchScript, а также инициализация системы озвучивания результатов (TextToSpeech). На этапе инициализации осуществляется запрос разрешения на доступ к камере, после чего запускается обработка видеопотока. Модель загружается из директории assets с помощью метода ModuleLoader.load(). [16]

private fun startApp() {

initializeTextToSpeech()

loadModel()

setupCamera()

}

...

private fun initializeTextToSpeech() {

textToSpeech = TextToSpeech(this, this)

}

private fun loadModel() {

executor.execute {

try {

val modelFile = File(filesDir, "model.pt")

if (!modelFile.exists()) {

assets.open("model.pt").use { input ->

modelFile.outputStream().use { output ->

input.copyTo(output)}}}

model = Module.load(modelFile.absolutePath)

...}

Для получения изображения используется CameraX, обеспечивающий удобный доступ к камере и поддержку потоковой передачи кадров. Настраивается Preview и ImageAnalysis, в котором каждые 500 миллисекунд осуществляется захват изображения и передача его в обработку.

Кадр, полученный из камеры (ImageProxy), преобразуется в формат Bitmap, пригодный для дальнейшей работы с PyTorch-моделью. Преобразование осуществляется с помощью YuvToRgbConverter и дальнейшей обрезки до квадратного изображения, соответствующего размеру, ожидаемому моделью.

После преобразования изображения в тензор формат (TensorImageUtils.bitmapToFloat32Tensor()), производится инференс с использованием предварительно загруженной модели. Выход модели интерпретируется как три задачи: [16]

1. **Классификация типа транспорта** — определение, является ли объект автобусом, троллейбусом, трамваем и т.п.
2. **Обнаружение координат номерного знака** — регрессия координат (x, y, width, height), описывающих ограничивающую рамку.
3. **Распознавание номера (OCR)** — посимвольное определение символов регистрационного знака.

В коде это реализовано путём разделения выходного тензора модели на сегменты, соответствующие каждой из задач.

Затем вычисляется метка класса с максимальным значением и расшифровывается строка номера на основе наибольших вероятностей для каждого символа.

private fun processWithModel(bitmap: Bitmap) {

try {

val inputTensor = TensorImageUtils.bitmapToFloat32Tensor(

bitmap,

TensorImageUtils.TORCHVISION\_NORM\_MEAN\_RGB,

TensorImageUtils.TORCHVISION\_NORM\_STD\_RGB)

val outputIValue = model?.forward(IValue.from(inputTensor))

...

val clsTensor = outputs[0].toTensor()

val bboxTensor = outputs[1].toTensor()

val ocrTensor = outputs[2].toTensor()

val classResult = interpretClassOutput(clsTensor.dataAsFloatArray)

val plateText = decodeOCROutput(ocrTensor)

val result = "Тип: $classResult\nНомер: $plateText"

...

runOnUiThread {

binding.tvResult.text = result

speak(result)}}

Полученные результаты отображаются на экране и одновременно озвучиваются для пользователя, что особенно важно для слабовидящих. Чтобы избежать повторного озвучивания одного и того же результата, используется переменная lastSpokenText, предотвращающая дублирование голосового вывода.

if (recognizedText != lastSpokenText) {

textToSpeech.speak(

recognizedText,

TextToSpeech.QUEUE\_FLUSH, null, null)

lastSpokenText = recognizedText}

Приложение реализует базовую обработку ошибок, включая защиту от непредвиденных исключений при инициализации камеры, загрузке модели и работе с TextToSpeech. При завершении работы приложения (onDestroy) освобождаются ресурсы и останавливается голосовой движок, предотвращая утечку памяти.

override fun onDestroy() {

textToSpeech.stop()

textToSpeech.shutdown()

super.onDestroy()

}

## 4.6 Тестирование и оценка эффективности системы

После завершения разработки системы были проведены два этапа тестирования: проверка качества работы обученной модели и оценка функционирования мобильного приложения.

Для оценки использовались случайные фотографии троллейбусов и синтетические тесты на отдельные части модели. В процессе тестирования выяснилось, что объём датасета оказалось недостаточным для качественного обучения. По результатам тестирования модель показала низкую точность. Правильное определение координат номерного знака (bounding box) происходило примерно в 20% случаев, а корректное распознавание текста — только в 10% изображений. Такие показатели указывают на существенные проблемы с объёмом обучающего датасета.

Отдельно было проведено тестирование мобильного приложения. Приложение стабильно запускалось, получало изображение с камеры, передавало его в модель и корректно отображало предсказания пользователю.

Это подтвердило, что архитектура приложения и его взаимодействие с нейросетевой моделью реализованы корректно. Несмотря на слабую точность самой модели, техническая часть приложения функционирует правильно.

Таким образом, тестирование показало, что архитектурно система работоспособна, но для достижения удовлетворительного качества распознавания необходимо доработать модель: увеличить и разнообразить датасет, улучшить качество изображений.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была реализована система, включающая в себя обучение нейронной сети и разработку Android-приложения, предназначенного для распознавания вида общественного транспорта и его маршрутного номера в условиях городской среды.

В рамках теоретической части были изучены методы компьютерного зрения и свёрточных нейронных сетей, рассмотрены существующие решения в области распознавания объектов и текста, а также проведён анализ инструментов, применимых для мобильной разработки. На основании этого анализа были выбраны оптимальные технологии: PyTorch для обучения модели, Kotlin, Android Studio и Gradle — для создания и сборки мобильного приложения, OpenCV — для обработки изображений.

Практическая часть включала сбор и разметку собственного датасета из 500 изображений различных видов транспорта. Разметка выполнялась в CVAT с указанием типа транспорта и области маршрутного номера. Для подготовки аннотаций к обучению был разработан специализированный скрипт конвертации.

Обученная модель была интегрирована в Android-приложение с использованием PyTorch Mobile. В приложении реализована захват и обработка видео с камеры, отображение предсказанного типа транспорта и распознанного номера маршрута. Интерфейс адаптирован для использования слабовидящими пользователями — с крупными элементами и озвучкой результатов.

В рамках финального этапа были проведены два вида тестирования: модели и приложения. Результаты показали, что несмотря на корректную техническую реализацию и стабильную работу мобильного приложения, сама модель продемонстрировала низкую точность. Bounding box номерного знака определялся правильно лишь в 20% случаев, а маршрутный номер — распознавался корректно примерно в 10% изображений. Причинами стали недостаточный объём и качество датасета.

Таким образом, разработанное решение подтверждает техническую реализуемость идеи и может рассматриваться как прототип. Однако для достижения практической эффективности требуется существенное улучшение модели: расширение и улучшение датасета, более тщательное обучение и валидация.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. **Twana Mustafa, Murat Karabatak. Challenges in Automatic License Plate Recognition System Review** [Электронный ресурс] : статья / IEEE Xplore. - URL: <https://ieeexplore.ieee.org/document/10131688> (дата обращения: 14.12.2024).
2. **Bengisu Dost.** How Can We Improve Microsoft's Seeing AI Application? [Электронный ресурс]: статья / Medium. - URL: <https://bengisudost.medium.com/how-can-we-improve-microsofts-seeing-ai-application-4cb4b074e0d6> (дата обращения: 14.12.2024).
3. **Timo Rohner. Python vs. C++ for Machine Learning** [Электронный ресурс]: статья / Netguru. - URL: <https://www.netguru.com/blog/cpp-vs-python> (дата обращения: 14.12.2024).
4. **Gabe Giro. Kotlin vs. Java: All-purpose Uses and Android Apps** [Электронный ресурс] : статья / Toptal. - URL: <https://www.toptal.com/kotlin/kotlin-vs-java> (дата обращения: 14.12.2024).
5. **Kurtis Pykes. PyTorch vs TensorFlow vs Keras** [Электронный ресурс]: статья / DataCamp. - URL: <https://www.datacamp.com/tutorial/pytorch-vs-tensorflow-vs-keras> (дата обращения: 14.12.2024).
6. **Getting Started with CameraX** [Электронный ресурс] : документация / Android Developers. - URL: <https://developer.android.com/codelabs/camerax-getting-started?hl=en#2> (дата обращения: 14.12.2024).
7. **Nadeem Yousuf Khanday, Shabir Ahmad Sofi. Object Recognition** [Электронный ресурс] : статья / ScienceDirect. - URL: <https://www.sciencedirect.com/topics/computer-science/object-recognition> (дата обращения: 18.12.2024).
8. Petru Potrimba. What is Optical Character Recognition (OCR)? [Электронный ресурс] : статья / Roboflow Blog. - URL: <https://blog.roboflow.com/what-is-optical-character-recognition-ocr/> (дата обращения: 18.12.2024).
9. **Howard, Gugger. Deep Learning for Coders with Fastai and PyTorch: AI Applications Without a PhD [Электронный ресурс] : книга** / Amazon. - URL: <https://www.amazon.com/Deep-Learning-Coders-fastai-PyTorch/dp/1492045527>
10. Alexander S. Gillis. What is a convolutional neural network (CNN)? [Электронный ресурс]: статья / TechTarget. - URL: <https://www.techtarget.com/searchenterpriseai/definition/convolutional-neural-network> (дата обращения: 18.12.2024).
11. **Zhang, Liptom. Dive into Deep Learning** [Электронный ресурс]: книга / Dive Into Deep Learning. - URL: <https://d2l.ai/> (дата обращения: 18.12.2024).
12. **Stevens, Antiga, Viehmann. Deep Learning with PyTorch** [Электронный ресурс]: книга/ ISIP. - URL: <https://isip.piconepress.com/courses/temple/ece_4822/resources/books/Deep-Learning-with-PyTorch.pdf> (дата обращения: 18.12.2024).
13. **CVAT Documentation** [Электронный ресурс] : официальная документация / CVAT.ai. - URL: <https://docs.cvat.ai/docs/> (дата обращения: 03.02.2025).
14. **PyTorch Stable Documentation** [Электронный ресурс] : официальная документация / PyTorch. - URL: <https://docs.pytorch.org/docs/stable/index.html> (дата обращения: 03.02.2025).
15. **Kotlin Documentation** [Электронный ресурс] : официальная документация / Kotlin Foundation. - URL: <https://kotlinlang.org/docs/home.html> (дата обращения: 20.12.2024).
16. **PyTorch ExecuTorch Documentation** [Электронный ресурс] : официальная документация / PyTorch. - URL: <https://docs.pytorch.org/executorch/stable/index.html> (дата обращения: 20.02.2025).

# ПРИЛОЖЕНИЕ А – Архитектура модели

multitask\_vehicle\_model.py

"""

Многозадачная модель для:

1. Классификации типа транспорта (5 классов)

2. Детекции bounding box номерного знака

3. Распознавания текста (OCR) с механизмом внимания

Архитектура:

- Общий CNN бэкбон для извлечения признаков

- Три специализированные "головы" для каждой задачи

- Механизм внимания для OCR

"""

import torch

import torch.nn as nn

import torch.nn.functional as F

import torch.nn.utils.rnn as rnn\_utils

class ConvBlock(nn.Module):

    """

    Базовый блок свертки с пулингом

    Состоит из:

    - Conv2d -> BatchNorm -> ReLU -> MaxPool2d

    Уменьшает размерность в 2 раза

    """

    def \_\_init\_\_(self, in\_channels, out\_channels, kernel\_size=3, stride=1, padding=1):

        super().\_\_init\_\_()

        self.block = nn.Sequential(

            nn.Conv2d(in\_channels, out\_channels, kernel\_size, stride, padding),

            nn.BatchNorm2d(out\_channels),  # Нормализация активаций

            nn.ReLU(inplace=True),        # Активация с оптимизацией памяти

            nn.MaxPool2d(2)               # Уменьшение размерности

        )

    def forward(self, x):

        return self.block(x)

class MultiTaskModel(nn.Module):

    """

    Многозадачная модель с тремя головками:

    1. Классификация транспорта

    2. Регрессия bounding box

    3. Распознавание текста (OCR)

    """

    def \_\_init\_\_(self, num\_classes=5, ocr\_vocab\_size=22 + 1, freeze\_cls=True):  # +1 для CTC blank

        super().\_\_init\_\_()

        self.num\_classes = num\_classes

        self.ocr\_vocab\_size = ocr\_vocab\_size

        self.freeze\_cls = freeze\_cls  # Флаг заморозки классификации

        # ------------------------------------------

        # 1. Общий бэкбон для извлечения признаков

        #    (оптимизирован под RTX 3060 12GB)

        # ------------------------------------------

        self.backbone = nn.Sequential(

            ConvBlock(3, 64),    # 512x512 -> 256x256

            ConvBlock(64, 128),   # 256x256 -> 128x128

            ConvBlock(128, 256),  # 128x128 -> 64x64

            ConvBlock(256, 512)   # 64x64 -> 32x32

        )

        # ------------------------------------------

        # 2. Головка классификации транспорта

        # ------------------------------------------

        self.cls\_head = nn.Sequential(

            nn.AdaptiveAvgPool2d((1, 1)),  # Глобальный пулинг

            nn.Flatten(),                  # Вытягивание в вектор

            nn.Linear(512, num\_classes)    # Финальная классификация

        )

        if freeze\_cls:

            # Закрепляем параметры классификации

            for param in self.cls\_head.parameters():

                param.requires\_grad = False

            # Фиксируем выход для одного класса

            self.register\_buffer('fixed\_cls\_output', torch.zeros(1, num\_classes).fill\_(-10.0))

            self.fixed\_cls\_output[0, 1] = 10.0  # Класс 1 (троллейбус)

        # ------------------------------------------

        # 3. Головка детекции bounding box

        #    (предсказывает [x\_center, y\_center, width, height])

        # ------------------------------------------

        self.bbox\_head = nn.Sequential(

            nn.Conv2d(512, 64, 3, padding=1),  # Локальные признаки

            nn.Flatten(),

            nn.Linear(64 \* 32 \* 32, 4),        # Регрессия координат

            nn.Sigmoid()                       # Нормализация в [0, 1]

        )

        # ------------------------------------------

        # 4. Головка OCR с механизмом внимания

        # ------------------------------------------

        #self.ocr\_pool = nn.AdaptiveAvgPool2d((1, None))  # Сохраняет ширину

        self.ocr\_feat\_proj = nn.Linear(512, 64)          # Проекция признаков

        self.ocr\_lstm = nn.LSTM(64, 32, batch\_first=True, bidirectional=False)

        # Механизм внимания

        self.ocr\_attention = nn.Sequential(

            nn.Linear(32, 1),  # Оценка важности каждого элемента последовательности

            nn.Softmax(dim=1)   # Нормализация весов

        )

        self.ocr\_fc = nn.Linear(32, ocr\_vocab\_size)  # Финальная классификация символов

    def forward(self, x):

        """

        Прямой проход через модель

        Возвращает кортеж:

        - cls\_out: логиты классов транспорта [B, num\_classes]

        - bbox\_out: координаты bbox [B, 4]

        - ocr\_out: логиты символов [B, 1, vocab\_size]

        """

        # 1. Извлечение общих признаков

        features = self.backbone(x)  # [B, 512, 32, 32]

        # 2. Классификация транспорта

        if self.freeze\_cls:

            # Возвращаем фиксированный выход для класса 1

            batch\_size = x.size(0)

            cls\_out = self.fixed\_cls\_output.repeat(batch\_size, 1)

        else:

            cls\_out = self.cls\_head(features)

        # 3. Детекция номера

        bbox\_out = self.bbox\_head(features)  # [B, 4]

        # 4. Распознавание текста

        # Подготовка признаков

        #ocr\_feat = self.ocr\_pool(features)  # [B, 512, 1, W]

        ocr\_feat = F.adaptive\_avg\_pool2d(features, (1, features.size(3)))  # [B, 512, 1, W]

        ocr\_feat = ocr\_feat.squeeze(2).permute(0, 2, 1)  # [B, W, 512]

        ocr\_feat = self.ocr\_feat\_proj(ocr\_feat)  # [B, W, 64]

        # Обработка LSTM

        ocr\_feat, \_ = self.ocr\_lstm(ocr\_feat)  # [B, W, 32]

        # Механизм внимания - не совместимо с TorchScript

        #attention\_weights = self.ocr\_attention(ocr\_feat)  # [B, W, 1]

        #attended\_features = (ocr\_feat \* attention\_weights).sum(dim=1)  # [B, 32]

        # Финальная классификация

        #ocr\_out = self.ocr\_fc(attended\_features).unsqueeze(1)  # [B, 1, vocab\_size]

        ocr\_out = self.ocr\_fc(ocr\_feat)                   # (B, W, vocab\_size)

        return cls\_out, bbox\_out, ocr\_out

    def compute\_losses(self, cls\_out, bbox\_out, ocr\_out, cls\_target, bbox\_target, ocr\_target, ocr\_lengths):

        """

        Вычисление всех функций потерь:

        1. Cross Entropy для классификации

        2. Smooth L1 Loss для регрессии bbox

        3. CTC Loss для OCR

        """

        # Явное приведение типов для совместимости

        cls\_out = cls\_out.float()

        bbox\_out = bbox\_out.float()

        ocr\_out = ocr\_out.float()

        # Проверка размерностей

        if cls\_out.dim() == 1:

            cls\_out = cls\_out.unsqueeze(0)

        cls\_target = cls\_target.long().view(-1)  # Приведение к [batch\_size]

        # Проверка совпадения размеров

        if cls\_out.shape[0] != cls\_target.shape[0]:

            raise ValueError(f"Shape mismatch: cls\_out {cls\_out.shape}, cls\_target {cls\_target.shape}")

        # Вычисление потерь

        if not self.freeze\_cls: # Классификация

            loss\_cls = F.cross\_entropy(cls\_out, cls\_target)

        else:

            loss\_cls = torch.tensor(0.0, device=cls\_out.device)

        #loss\_bbox = F.smooth\_l1\_loss(bbox\_out, bbox\_target)  # Регрессия

        loss\_bbox = F.mse\_loss(bbox\_out, bbox\_target) \* 10 # Регрессия

        loss\_ocr = self.ocr\_ctc\_loss(ocr\_out, ocr\_target, ocr\_lengths)  # OCR

        return loss\_cls + loss\_bbox + loss\_ocr, loss\_cls, loss\_bbox, loss\_ocr

    def ocr\_ctc\_loss(self, ocr\_out, ocr\_target, target\_lengths):

        """

        Connectionist Temporal Classification (CTC) Loss

        с явным приведением к float32 для совместимости

        Args:

            ocr\_out: Выход модели [B, 1, vocab\_size]

            ocr\_target: Целевые последовательности [B, max\_len]

            target\_lengths: Длины целевых последовательностей [B]

        """

        # Приведение к float32 для CTC loss

        ocr\_out = ocr\_out.float()  # Явное преобразование

        target\_lengths = target\_lengths.to(ocr\_out.device)

        # Проверка размерностей

        if ocr\_out.dim() != 3:

            raise ValueError(f"ocr\_out должен быть 3D тензором, получено {ocr\_out.shape}")

        if target\_lengths.dim() != 1:

            raise ValueError(f"target\_lengths должен быть 1D тензором, получено {target\_lengths.shape}")

        log\_probs = ocr\_out.log\_softmax(dim=2)  # [B, 1, vocab\_size]

        log\_probs = log\_probs.permute(1, 0, 2)  # [1, B, vocab\_size]

        # Убедимся, что target\_lengths имеет правильный размер

        if len(target\_lengths) != log\_probs.size(1):

            raise ValueError(

                f"Несоответствие размеров: target\_lengths {len(target\_lengths)} != batch\_size {log\_probs.size(1)}"

            )

        input\_lengths = torch.full(

            size=(log\_probs.size(1),),

            fill\_value=log\_probs.size(0),

            dtype=torch.long,

            device=log\_probs.device

        )

        return F.ctc\_loss(

            log\_probs=log\_probs,

            targets=ocr\_target,

            input\_lengths=input\_lengths,

            target\_lengths=target\_lengths,

            blank=self.ocr\_vocab\_size - 1,

            zero\_infinity=True

        )

    def predict(self, x, cls\_threshold=0.6, ocr\_confidence\_threshold=0.4):

        """

        Инференс с постобработкой

        Возвращает:

        - pred\_class: предсказанный класс транспорта

        - pred\_bbox: координаты bounding box [x, y, w, h]

        - pred\_text: распознанный текст

        """

        # Прямой проход

        cls\_out, bbox\_out, ocr\_out = self.forward(x)

        # 1. Обработка классификации

        cls\_probs = F.softmax(cls\_out, dim=1)

        cls\_pred = torch.argmax(cls\_probs, dim=1)

        # Помечаем как "неизвестный" при низкой уверенности

        cls\_pred[cls\_probs[torch.arange(len(cls\_probs)), cls\_pred] < cls\_threshold] = 4

        # 2. Фильтрация bbox

        plate\_bbox = bbox\_out if (bbox\_out[:, 2] \* bbox\_out[:, 3] > 0.01).all() else None

        # 3. Декодирование OCR

        ocr\_probs = F.softmax(ocr\_out, dim=2)

        top\_char = torch.argmax(ocr\_probs, dim=2)

        top\_conf = torch.max(ocr\_probs, dim=2).values

        if top\_conf.item() < ocr\_confidence\_threshold:

            plate\_text = "Не распознано"

        else:

            plate\_text = self.decode\_plate(top\_char[0])

        return cls\_pred, plate\_bbox, plate\_text

    def decode\_plate(self, char\_indices):

        """Преобразование индексов в строку с фильтрацией blank символов"""

        chars = "0123456789ABEKMHOPCTYX"  # Словарь символов

        return ''.join([chars[i] for i in char\_indices if i < len(chars)])

# ****ПРИЛОЖЕНИЕ Б – Обучение модели****

**train.py**

"""

Модуль для обучения многозадачной модели:

1. Классификация типа транспорта

2. Детекция bounding box

3. Распознавание текста (OCR)

Основные компоненты:

- Загрузка и обработка данных

- Инициализация модели

- Цикл обучения с mixed precision

- Логирование и сохранение моделей

"""

import torch

from torch.utils.data import DataLoader

from torch.optim import Adam, lr\_scheduler

from torch.nn.utils import clip\_grad\_norm\_

from dataset import VehicleDataset, collate\_fn

from multitask\_vehicle\_model import MultiTaskModel

import yaml

from tqdm import tqdm  # Для красивого прогресс-бара

import os

import numpy as np

# ==============================================

# Основная функция обучения

# ==============================================

def main():

    """

    Главный цикл обучения модели. Выполняет:

    1. Загрузку конфигурации

    2. Подготовку данных

    3. Инициализацию модели

    4. Цикл обучения с валидацией

    5. Сохранение лучшей модели

    """

    # ------------------------------

    # 1. Конфигурация и настройки

    # ------------------------------

    # Загрузка параметров из YAML-файла

    with open("config.yaml") as f:

        config = yaml.safe\_load(f)

    # Автоматический выбор устройства (GPU/CPU)

    device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

    # ------------------------------

    # 2. Подготовка модели

    # ------------------------------

    # Инициализация многозадачной модели

    model = MultiTaskModel(

        num\_classes=5,  # Количество классов транспорта

        ocr\_vocab\_size=len(config['model']['vocab']) + 1, # Размер словаря OCR + blank символ

        freeze\_cls=True # Замораживаем классификацию

    ).to(device)  # Перенос модели на выбранное устройство

    # Отключаем autocast для CTC loss

    @torch.autocast(device\_type=device.type, enabled=False)

    def compute\_losses\_wrapper(\*args, \*\*kwargs):

        return model.compute\_losses(\*args, \*\*kwargs)

    # ------------------------------

    # 3. Загрузка данных

    # ------------------------------

    # Создание датасета для обучения

    train\_dataset = VehicleDataset(

        root\_dir="E:\IT\data",  # Путь к обучающим данным

        vocab=config['model']['vocab'],  # Словарь символов

        image\_size=config['image']['size'],  # Размер входного изображения

        augment=True # Аугментация данных

    )

    # Создание DataLoader для батчевой обработки

    train\_loader = DataLoader(

        train\_dataset,

        batch\_size=config['training']['batch\_size'],  # Размер батча

        shuffle=True,  # Перемешивание данных

        collate\_fn=collate\_fn,  # Функция для сборки батчей

        num\_workers=0,  # Количество процессов для загрузки (0 для Windows)

        pin\_memory=True  # Ускоряет перенос данных на GPU

    )

    # ------------------------------

    # 4. Оптимизация и планировщик

    # ------------------------------

    # Инициализация оптимизатора Adam

    optimizer = Adam(

        model.parameters(),  # Параметры для оптимизации

        lr=config['training']['lr']  # Скорость обучения

    )

    # Планировщик для динамического изменения lr

    scheduler = lr\_scheduler.ReduceLROnPlateau(

        optimizer,

        mode='min',  # Мониторим уменьшение потерь

        factor=0.5,  # Коэффициент уменьшения lr

        patience=3  # Количество эпох без улучшения перед уменьшением lr

    )

    # Инициализация GradScaler для mixed precision обучения

    scaler = torch.amp.GradScaler(

        device='cuda' if torch.cuda.is\_available() else 'cpu'

    )

    # ------------------------------

    # 5. Цикл обучения

    # ------------------------------

    best\_loss = float('inf')  # Лучший показатель потерь

    for epoch in range(config['training']['epochs']):

        model.train()  # Переводим модель в режим обучения

        epoch\_loss = 0.0  # Суммарные потери за эпоху

        # Прогресс-бар для визуализации обучения

        progress\_bar = tqdm(

            train\_loader,

            desc=f"Epoch {epoch+1}/{config['training']['epochs']}"

        )

        for batch in progress\_bar:

            # Перенос данных на устройство (GPU/CPU)

            images, cls\_targets, bbox\_targets, ocr\_targets, ocr\_lengths = [

                x.to(device, non\_blocking=True) for x in batch

            ]

            # Обнуление градиентов (оптимизированная версия)

            optimizer.zero\_grad(set\_to\_none=True)

            # Forward pass с mixed precision

            with torch.amp.autocast(device\_type=device.type):

                # Получение предсказаний модели

                cls\_out, bbox\_out, ocr\_out = model(images)

                # Вычисление функции потерь

                loss, loss\_cls, loss\_bbox, loss\_ocr = model.compute\_losses(

                    cls\_out,  # Выход классификатора

                    bbox\_out,  # Выход детектора bbox

                    ocr\_out,  # Выход OCR

                    cls\_targets.view(-1),  # Целевые классы (приводим к [batch\_size])

                    bbox\_targets,  # Целевые bbox

                    ocr\_targets,  # Целевые тексты

                    ocr\_lengths  # Длины текстов

                )

            # Backward pass с масштабированием градиентов

            scaler.scale(loss).backward()

            # Обрезка градиентов для стабильности

            clip\_grad\_norm\_(model.parameters(), 1.0)

            # Шаг оптимизации

            scaler.step(optimizer)

            # Обновление масштаба

            scaler.update()

            # Логирование

            epoch\_loss += loss.item()

            progress\_bar.set\_postfix({

                "loss": f"{loss.item():.4f}",

                "cls": f"{loss\_cls.item():.4f}",

                "bbox": f"{loss\_bbox.item():.4f}",

                "ocr": f"{loss\_ocr.item():.4f}"

            })

        # ------------------------------

        # 6. Валидация и сохранение

        # ------------------------------

        avg\_loss = epoch\_loss / len(train\_loader)

        # Обновление learning rate

        scheduler.step(avg\_loss)

        # Сохранение лучшей модели

        if avg\_loss < best\_loss:

            best\_loss = avg\_loss

            torch.save({

                'epoch': epoch,

                'model\_state\_dict': model.state\_dict(),

                'optimizer\_state\_dict': optimizer.state\_dict(),

                'loss': best\_loss,

            }, "best\_model.pth")

        print(f"Epoch {epoch+1} | Train Loss: {avg\_loss:.4f} | LR: {optimizer.param\_groups[0]['lr']:.2e}")

# Точка входа (особенно важна для Windows)

if \_\_name\_\_ == "\_\_main\_\_":

main()

# ПРИЛОЖЕНИЕ В – Загрузка датасета

dataset.py

"""

Модуль для обучения многозадачной модели:

1. Классификация типа транспорта

2. Детекция bounding box

3. Распознавание текста (OCR)

Основные компоненты:

- Загрузка и обработка данных

- Инициализация модели

- Цикл обучения с mixed precision

- Логирование и сохранение моделей

"""

import torch

from torch.utils.data import DataLoader

from torch.optim import Adam, lr\_scheduler

from torch.nn.utils import clip\_grad\_norm\_

from dataset import VehicleDataset, collate\_fn

from multitask\_vehicle\_model import MultiTaskModel

import yaml

from tqdm import tqdm  # Для красивого прогресс-бара

import os

import numpy as np

# ==============================================

# Основная функция обучения

# ==============================================

def main():

    """

    Главный цикл обучения модели. Выполняет:

    1. Загрузку конфигурации

    2. Подготовку данных

    3. Инициализацию модели

    4. Цикл обучения с валидацией

    5. Сохранение лучшей модели

    """

    # ------------------------------

    # 1. Конфигурация и настройки

    # ------------------------------

    # Загрузка параметров из YAML-файла

    with open("config.yaml") as f:

        config = yaml.safe\_load(f)

    # Автоматический выбор устройства (GPU/CPU)

    device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

    # ------------------------------

    # 2. Подготовка модели

    # ------------------------------

    # Инициализация многозадачной модели

    model = MultiTaskModel(

        num\_classes=5,  # Количество классов транспорта

        ocr\_vocab\_size=len(config['model']['vocab']) + 1, # Размер словаря OCR + blank символ

        freeze\_cls=True # Замораживаем классификацию

    ).to(device)  # Перенос модели на выбранное устройство

    # Отключаем autocast для CTC loss

    @torch.autocast(device\_type=device.type, enabled=False)

    def compute\_losses\_wrapper(\*args, \*\*kwargs):

        return model.compute\_losses(\*args, \*\*kwargs)

    # ------------------------------

    # 3. Загрузка данных

    # ------------------------------

    # Создание датасета для обучения

    train\_dataset = VehicleDataset(

        root\_dir="E:\IT\data",  # Путь к обучающим данным

        vocab=config['model']['vocab'],  # Словарь символов

        image\_size=config['image']['size'],  # Размер входного изображения

        augment=True # Аугментация данных

    )

    # Создание DataLoader для батчевой обработки

    train\_loader = DataLoader(

        train\_dataset,

        batch\_size=config['training']['batch\_size'],  # Размер батча

        shuffle=True,  # Перемешивание данных

        collate\_fn=collate\_fn,  # Функция для сборки батчей

        num\_workers=0,  # Количество процессов для загрузки (0 для Windows)

        pin\_memory=True  # Ускоряет перенос данных на GPU

    )

    # ------------------------------

    # 4. Оптимизация и планировщик

    # ------------------------------

    # Инициализация оптимизатора Adam

    optimizer = Adam(

        model.parameters(),  # Параметры для оптимизации

        lr=config['training']['lr']  # Скорость обучения

    )

    # Планировщик для динамического изменения lr

    scheduler = lr\_scheduler.ReduceLROnPlateau(

        optimizer,

        mode='min',  # Мониторим уменьшение потерь

        factor=0.5,  # Коэффициент уменьшения lr

        patience=3  # Количество эпох без улучшения перед уменьшением lr

    )

    # Инициализация GradScaler для mixed precision обучения

    scaler = torch.amp.GradScaler(

        device='cuda' if torch.cuda.is\_available() else 'cpu'

    )

    # ------------------------------

    # 5. Цикл обучения

    # ------------------------------

    best\_loss = float('inf')  # Лучший показатель потерь

    for epoch in range(config['training']['epochs']):

        model.train()  # Переводим модель в режим обучения

        epoch\_loss = 0.0  # Суммарные потери за эпоху

        # Прогресс-бар для визуализации обучения

        progress\_bar = tqdm(

            train\_loader,

            desc=f"Epoch {epoch+1}/{config['training']['epochs']}"

        )

        for batch in progress\_bar:

            # Перенос данных на устройство (GPU/CPU)

            images, cls\_targets, bbox\_targets, ocr\_targets, ocr\_lengths = [

                x.to(device, non\_blocking=True) for x in batch

            ]

            # Обнуление градиентов (оптимизированная версия)

            optimizer.zero\_grad(set\_to\_none=True)

            # Forward pass с mixed precision

            with torch.amp.autocast(device\_type=device.type):

                # Получение предсказаний модели

                cls\_out, bbox\_out, ocr\_out = model(images)

                # Вычисление функции потерь

                loss, loss\_cls, loss\_bbox, loss\_ocr = model.compute\_losses(

                    cls\_out,  # Выход классификатора

                    bbox\_out,  # Выход детектора bbox

                    ocr\_out,  # Выход OCR

                    cls\_targets.view(-1),  # Целевые классы (приводим к [batch\_size])

                    bbox\_targets,  # Целевые bbox

                    ocr\_targets,  # Целевые тексты

                    ocr\_lengths  # Длины текстов

                )

            # Backward pass с масштабированием градиентов

            scaler.scale(loss).backward()

            # Обрезка градиентов для стабильности

            clip\_grad\_norm\_(model.parameters(), 1.0)

            # Шаг оптимизации

            scaler.step(optimizer)

            # Обновление масштаба

            scaler.update()

            # Логирование

            epoch\_loss += loss.item()

            progress\_bar.set\_postfix({

                "loss": f"{loss.item():.4f}",

                "cls": f"{loss\_cls.item():.4f}",

                "bbox": f"{loss\_bbox.item():.4f}",

                "ocr": f"{loss\_ocr.item():.4f}"

            })

        # ------------------------------

        # 6. Валидация и сохранение

        # ------------------------------

        avg\_loss = epoch\_loss / len(train\_loader)

        # Обновление learning rate

        scheduler.step(avg\_loss)

        # Сохранение лучшей модели

        if avg\_loss < best\_loss:

            best\_loss = avg\_loss

            torch.save({

                'epoch': epoch,

                'model\_state\_dict': model.state\_dict(),

                'optimizer\_state\_dict': optimizer.state\_dict(),

                'loss': best\_loss,

            }, "best\_model.pth")

        print(f"Epoch {epoch+1} | Train Loss: {avg\_loss:.4f} | LR: {optimizer.param\_groups[0]['lr']:.2e}")

if \_\_name\_\_ == "\_\_main\_\_":

    main()