[1. ПАСПОРТ ПРОЕКТА](#_hrsrzcjscnjw)

[2. ОБЗОРНАЯ ЧАСТЬ](#_a4un9f9sn9lc)

[2.1. Обзор задачи сегментации объектов на изображениях](#_41xpaqfvjgde)

[📊 Сравнительная таблица типов сегментации изображений](#_e9cs63ydh33x)

[2.2. Обзор алгоритмов машинного обучения и архитектур нейронных сетей для сегментации](#_2nu8gflu5p7t)

[📊 Сравнительная таблица архитектур для семантической сегментации](#_3lfwwkszpy5d)

[3. РАЗРАБОТКА СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА](#_6y18amhs90zk)

[3.1. Постановка задачи](#_17hmaokxvzz)

[Целевые метрики](#_g2instlfe6fy)

[3.2. Описание и анализ датасета](#_27r5cr768raz)

[Используемый основной датасет](#_utdrqappiyql)

[Анализ классов и баланса](#_v86dae9vebyw)

[Обоснование обобщаемости](#_dw30ciajt36x)

[3.3. Подготовка данных](#_9mi0x9vram0b)

[3.4. Архитектура сетей](#_ya7vwemng6yy)

[3.4.1. Общая схема ESNet](#_d9e0ijeac71)

[3.4.2. Efficient Residual Module (ERM)](#_56xh20dfn4jr)

[3.4.3. Lightweight Context Module (LCM)](#_hogggoojiho8)

[3.4.4. Pyramid Pooling Module (PPM)](#_zruivzf3kod)

[3.4.5. Реализация в PyTorch](#_ghfhq22rkt3c)

[3.4.6. Оптимизация и экспорт](#_zgfzrxmvi6p2)

[4. ВЫЧИСЛИТЕЛЬНЫЕ ЭКСПЕРИМЕНТЫ](#_992nak7m31e5)

[4.1. Метод обучения](#_5n1yxs996nzi)

[4.2. Оценка качества модели](#_ju8afha90bbo)

[4.2. Оценка качества модели](#_9l74o0zg5rgk)

[4.3. Анализ качества и ошибок](#_i7rl9yq71xws)

[Сильные стороны](#_dmtyzmiltbgb)

[Ошибки и слабые классы](#_mzia9q265lxr)

[Интерпретируемость](#_nbblkm8z8xhj)

[4.4. Применение обученной модели в реальной среде](#_7qbe1skhw2o6)

[Интеграция в Android-приложение](#_v1jqdgnom1e0)

[Тестирование в условиях города](#_yk8yfurlq5s6)

[Архитектура передачи](#_savs24n5yuha)

[5. ИНФРАСТРУКТУРА И РЕАЛИЗАЦИЯ](#_gmvwferv98u6)

[5.1. Docker-окружение](#_3c0e0ohbdy5t)

[5.2. Android-клиент](#_ikxujdrgntl)

[6. ОСНОВНЫЕ ЭТАПЫ РЕАЛИЗАЦИИ](#_ykjg5bhwj4xq)

[6.1. Индивидуальные задачи по этапам проекта](#_mz11ihh9e0nu)

[7. ЗАКЛЮЧЕНИЕ](#_xphnw8cf2don)

[8. БИБЛИОГРАФИЧЕСКИЙ СПИСОК](#_6uacjjd2wq8u)

[Наши ссылки:](#_l6647h4vs9yn)

## **1. ПАСПОРТ ПРОЕКТА**

**Описание задачи:** Разработка системы для семантической сегментации городских сцен в реальном времени с использованием нейронной сети. Предусматривается создание Android-приложения, позволяющего отправлять фотографии на сервер, где модель выполняет сегментацию, а результат возвращается пользователю. Модель развёрнута в Docker-контейнере для обеспечения удобства и переносимости.

**Цель работы:** Разработать систему реального времени для сегментации объектов на изображениях городских сцен с использованием моделей машинного обучения и реализовать их интеграцию в мобильное приложение.

**Задачи:**

* провести обзор существующих решений и моделей для семантической сегментации изображений;
* обосновать выбор модели ESNet для задачи сегментации в реальном времени;
* собрать и подготовить набор данных, обеспечивающий обобщаемость модели;
* обучить модель ESNet на выбранном датасете;
* развернуть модель в Docker-контейнере для серверного исполнения;
* разработать Android-приложение для отправки изображений и отображения результата сегментации;
* протестировать систему в реальных условиях (городские сцены, фото с мобильного устройства).

**Актуальность и практическая значимость:** Семантическая сегментация городских сцен находит применение в системах умного города, навигации, автономном вождении и мониторинге городской инфраструктуры. Модель, работающая в реальном времени и интегрированная с мобильным приложением, может использоваться для анализа объектов на фото без необходимости дорогого оборудования, что расширяет её прикладное значение для широкого круга пользователей.

**Ожидаемые результаты:**

* обученная модель ESNet, способная производить сегментацию городских сцен в реальном времени;
* Docker-сервис с REST API для обработки изображений;
* мобильное Android-приложение с интерфейсом для загрузки изображений и отображения результата;
* отчёт о точности модели и тестировании в условиях, приближённых к реальной эксплуатации.

**Инструменты и технологии:**

* Язык программирования: Python,Kotlin
* Фреймворки: PyTorch, TensorFlow Lite, OpenCV, Compose (UI adndoid)
* Модель: ESNet
* Инфраструктура: Docker, Flask или FastAPI
* Среда разработки: Google Colab, Android Studio
* Датасеты: Cityscapes, driving-segmentation, а также дополнительные наборы (сформированные нашей командой)

## **2. ОБЗОРНАЯ ЧАСТЬ**

### **2.1. Обзор задачи сегментации объектов на изображениях**

Сегментация объектов на изображениях — это ключевая задача в области компьютерного зрения, направленная на выделение объектов или областей на изображении с пиксельной точностью. В зависимости от целей, сегментация делится на несколько типов: бинарную, многоклассовую, инстанс-сегментацию и **семантическую сегментацию**. Последняя представляет наибольший интерес в рамках данной работы, так как позволяет классифицировать каждый пиксель изображения в соответствии с его принадлежностью к определённому классу (дорога, автомобиль, пешеход и т.д.).

### **📊 Сравнительная таблица типов сегментации изображений**

| **Тип сегментации** | **Описание** | **Количество классов** | **Различие между объектами одного класса** | **Пример применения** |
| --- | --- | --- | --- | --- |
| **Бинарная сегментация** | Делит изображение на объект и фон. | 2 (объект / фон) | Нет | Выделение опухолей на МРТ, обнаружение людей на фоне |
| **Многоклассовая сегментация** | Классифицирует пиксели по нескольким классам. | >2 (каждый пиксель — 1 класс) | Нет | Сегментация дорог, зданий, деревьев и пр. |
| **Семантическая сегментация** | Классифицирует каждый пиксель в определённый класс, без различия между объектами. | >2 (например, «машина», «пешеход») | Нет (все объекты одного класса объединены) | Городская сцена: дороги, автомобили, пешеходы |
| **Инстанс-сегментация** | Комбинирует семантическую сегментацию с детектированием отдельных объектов. | >2 | Да (каждый объект размечен отдельно) | Подсчёт машин на парковке, отслеживание пешеходов в толпе |

Таблица-сравнение четырёх типов сегментации изображений: бинарной, многоклассовой, семантической и инстанс-сегментации

Семантическая сегментация активно применяется в следующих сферах:

* системы автономного вождения и помощи водителю (ADAS),
* анализ городской инфраструктуры и мониторинг состояния дорог,
* видеонаблюдение и обеспечение безопасности,
* мобильные AR/VR-приложения и навигационные системы.

Одной из ключевых задач при разработке сегментирующих систем является достижение баланса между точностью и производительностью. Особенно это актуально в мобильных и real-time-приложениях, где ресурсы ограничены, а задержки недопустимы. В таких условиях особую важность приобретают лёгкие, но эффективные архитектуры, способные обрабатывать изображения в реальном времени.

Однако реализация таких решений сталкивается с рядом сложностей:

* **разнообразие и сложность городской среды**, включая различия в архитектуре, погодных условиях и освещении;
* **ограниченные вычислительные ресурсы** на клиентских устройствах (например, в смартфонах);
* **необходимость в обобщаемости**, особенно при использовании открытых датасетов, собранных в других странах и условиях.

Для преодоления этих проблем необходимо использовать как мощные, так и оптимизированные модели нейронных сетей, обеспечивающие высокое качество при умеренной нагрузке на систему.

### **2.2. Обзор алгоритмов машинного обучения и архитектур нейронных сетей для сегментации**

Современные решения в области семантической сегментации строятся преимущественно на основе сверточных нейронных сетей (CNN) и их производных. Среди наиболее известных архитектур можно выделить:

* **FCN (Fully Convolutional Networks)** — первая архитектура, использующая только сверточные слои для пиксельной классификации. Однако модель имеет недостаточную точность на границах объектов.
* **U-Net** — популярная архитектура с симметричной структурой энкодера-декодера, применяемая в медицинской сегментации. Хорошо восстанавливает пространственное разрешение.
* **DeepLab (v1–v3+)** — серия моделей с использованием дырчатых (dilated) свёрток и модулей Atrous Spatial Pyramid Pooling для захвата контекста на разных масштабах.
* **PSPNet (Pyramid Scene Parsing Network)** — предлагает пирамидальное представление контекста сцены, что особенно полезно в городской среде.
* **BiSeNet** — раздельно обрабатывает пространственную и контекстную информацию, обеспечивая высокую скорость при сохранении качества.

### 

### **📊 Сравнительная таблица архитектур для семантической сегментации**

| **Архитектура** | **Плюсы** | **Минусы** | **Заключение по применимости в проекте** |
| --- | --- | --- | --- |
| **FCN** (Fully Convolutional Networks) | - Простая реализация  - Лёгкая по ресурсам  - Основа для других моделей | - Низкая точность  - Слабое восстановление границ объектов | ❌ Слишком базовая для задач городской сегментации |
| **U-Net** | - Хорошее восстановление форм  - Устойчивость при обучении  - Простота адаптации под разные задачи | - Большой объём параметров  - Не рассчитана на городские сцены  - Слабая оптимизация под реальное время | ⚠️ Подходит только в случае мощного оборудования и офлайн-обработки |
| **DeepLab (v1–v3+)** | - Отличное извлечение контекста  - Высокая точность  - Совместимость с предобученными энкодерами | - Высокие вычислительные затраты  - Низкая скорость на слабых устройствах | ⚠️ Надёжная, но малопригодна для мобильного real-time применения |
| **PSPNet** | ⭐ Максимально полное понимание сцены за счёт пирамидального контекста  ⭐ Высокая точность на городских датасетах  ⭐ Устойчива к различным условиям (освещение, плотность объектов)  ⭐ Подходит для сложных реальных сценариев | - Относительно высокая ресурсоёмкость  - Необходима оптимизация под мобильные устройства | ✅ Идеальна для качественной сегментации в городских условиях, особенно при Docker-обработке или серверной части приложения |
| **BiSeNet** | - Очень высокая скорость  - Хороший компромисс между точностью и производительностью  - Подходит для мобильных устройств | - Уступает по точности PSPNet и DeepLab  - Менее устойчива к сложным сценам | ⚠️ Отлично для лёгких real-time приложений, но проигрывает по глубине анализа сцен |

Сравнение архитектуры нейронных сетей для сегментации

В рамках данной работы будет рассматривается модель **ESNet (Efficient Scene Segmentation Network)** — современная архитектура, спроектированная специально для задач сегментации в реальном времени. Ключевые особенности ESNet:

* модульная структура с использованием MobileNet-блоков;
* высокая эффективность и малое количество параметров;
* возможность обработки изображений со скоростью до 60 FPS на мобильных устройствах;
* хорошая точность на городских датасетах (Cityscapes).

ESNet представляет собой сбалансированное решение между производительностью и качеством сегментации, что делает её особенно подходящей для использования в Android-приложениях, ориентированных на работу в реальных условиях городской среды.

Помимо архитектур сегментации, существует направление моделей объектного детектирования, таких как YOLO (You Only Look Once), широко применяемое для задач обнаружения объектов. Однако в рамках данной работы задача заключается не просто в обнаружении курьера, а в **точной пиксельной сегментации городской сцены**, включая разметку дороги, тротуаров, зданий, неба и пр. YOLO не обеспечивает такого уровня детализации и контекстной информации. Кроме того, формат обучающих данных (Cityscapes, driving-segmentation) предполагает наличие сегментационных масок, а не ограничивающих рамок. В связи с этим архитектуры сегментации (в частности, ESNet) являются предпочтительным выбором.

YOLO предсказывает прямоугольники (bounding boxes), а не маски. А задача исследования — пиксельная сегментация городской сцены, где важны границы, формы и площади объектов. Также YOLO не умеет различать классы на уровне пикселей. Например, он скажет: «в этом прямоугольнике курьер», но не скажет, какие конкретно пиксели на изображении принадлежат курьеру, а какие — скутеру или дороге.

| **Критерий** | **YOLO** | **ESNet / PSPNet / DeepLab (сегментация)** |
| --- | --- | --- |
| **Тип задачи** | Обнаружение объектов (bounding boxes) | Семантическая сегментация (mask per pixel) |
| **Выход модели** | Координаты и классы объектов | Маска: класс каждого пикселя |
| **Подходит ли для задач типа "где дорога, тротуар, небо"?** | ❌ Нет | ✅ Да |
| **Подходит ли для анализа городской сцены?** | ⚠️ Частично (только bbox объектов) | ✅ Полностью |
| **Можно ли детектировать курьеров?** | ✅ Да (но только прямоугольники) | ✅ Да, с точным контуром |
| **Подходит ли для Android?** | ✅ Да, с оптимизациями | ✅ Да, если архитектура лёгкая (например, ESNet) |

Краткое сравнение: YOLO vs Сегментационные модели

## 

## **3. РАЗРАБОТКА СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА**

### **3.1. Постановка задачи**

**Формализация задачи семантической сегментации городских сцен с учётом курьеров**

Цель — построить модель, способную выполнять семантическую сегментацию городских изображений в условиях реального времени. Основное нововведение — дообучение модели на кастомных данных, содержащих курьеров на скутерах и велосипедах, с целью корректной классификации и различения этих объектов от других участников движения.

Модель обучается на размеченных данных, где каждому пикселю соответствует правильная метка. Основной вызов заключается в достижении хорошего качества сегментации при сохранении высокой скорости обработки — особенно важно для приложений в реальном времени.

#### **Целевые метрики**

Целевые метрики включают:

* Потери Dice Loss для многоклассовой сегментации, которая хорошо подходит для оценки качества сегментации с учётом перекрытия предсказанных и истинных масок.
* Дополнительно могут использоваться метрики точности, полноты и среднее пересечение (mIoU) для оценки качества сегментации по каждому классу.

### **3.2. Описание и анализ датасета**

#### **Используемый основной датасет**

В качестве основного набора данных был выбран открытый датасет **Cityscapes**, ориентированный на задачи сегментации городских сцен. Он содержит высококачественные аннотированные изображения, снятые с автомобильной камеры в различных городах Европы. Каждый снимок имеет соответствующую маску с 30+ классами объектов городской инфраструктуры.

**📊 Таблица сравнения доступных пакетов Cityscapes**

| **Пакет** | **Описание** | **Размер** | **Применимость** | **Преимущества** | **Недостатки** | **Вывод** |
| --- | --- | --- | --- | --- | --- | --- |
| **gtFine\_trainvaltest.zip** | Точные аннотации (тонкая разметка) для train/val, пустые маски для test | 241MB | ✅ Обучение и валидация | Высокое качество масок, подходит для supervised обучения | Только 3475 изображений с масками | **Нужен обязательно** |
| **leftImg8bit\_trainvaltest.zip** | 8-бит изображения, соответствующие gtFine | 11GB | ✅ Входные данные для модели | Соответствуют аннотациям, хорошее качество | Ограниченное количество изображений (5000) | **Нужен обязательно** |
| gtCoarse.zip | Грубые аннотации + train\_extra | 1.3GB | ⚠️ Ограниченно | Больше данных, подходит для pretraining | Качество разметки хуже, не для финального обучения | ❌ Не обязателен |
| leftImg8bit\_trainextra.zip | 20K дополнительных изображений | 44GB | ⚠️ Опционально | Много данных для pretraining/unsupervised | Нет точных аннотаций (gtFine) | ❌ Не нужен |
| rightImg8bit\_trainvaltest.zip | Правые изображения для стерео | 11GB | ❌ Не актуально | Для 3D- или depth-анализа | Не используются в 2D-сегментации | ❌ Не нужен |
| disparity\_trainvaltest.zip | Карты глубины (SGM) | 3.5GB | ❌ Не актуально | Для depth-aware сегментации | Требуют моделей, учитывающих глубину | ❌ Не нужен |
| leftImg8bit\_blurred.zip | Лицензии и лица размыты | 11GB | ❌ Только для визуализации | GDPR-friendly | Не использовать для inference | ❌ Не нужен |
| leftImg8bit\_sequence\_trainvaltest.zip | Видео-сегменты (30 кадров) | 324GB | ❌ Только для видео-анализа | Подходит для видео-сегментации | Огромный размер, не для real-time single frame | ❌ Не нужен |
| leftImg16bit\_trainvaltest.zip | HDR-изображения | 37GB | ❌ Избыточно | Повышенный динамический диапазон | Требует переработки пайплайна | ❌ Не нужен |

#### 

#### **Анализ классов и баланса**

Датасет содержит как часто встречающиеся классы (дорога, автомобили, здания), так и менее распространённые (велосипедисты, светофоры). Классы имеют сильный **дисбаланс**: пиксели, соответствующие дороге или зданиям, доминируют над более мелкими объектами, что требует использования взвешенных потерь при обучении или специализированных техник усиления малых классов.

#### **Обоснование обобщаемости**

Хотя Cityscapes создавался в европейских городах, базовая инфраструктура (дороги, тротуары, пешеходы, машины) во многом соответствует российским условиям. Для повышения обобщающей способности модели планируется дополнительно использовать изображения с других открытых датасетов, таких как набора driving-segmentation, а также **добавить фотографии из локальной инфраструктуры**, снятые на мобильное устройство и размеченные вручную. Это поможет адаптировать модель под региональные особенности (тип дорожной разметки, стиль зданий, вывески и т.д.).

**Кастомный датасет (кастомный параметр “курьеры”):**Для повышения релевантности модели в условиях российской городской среды и детектирования курьеров были собраны изображения курьеров на мопедах и велосипедах, снятые в городских условиях (с мобильного телефона и с камеры Sony Alpha). Эти данные размечены вручную с использованием open-source инструментов (LabelMe) и включены в датасет. Это расширение позволило повысить обобщаемость модели и сместить её от зависимости от англоязычных датасетов. Добавился новый класс: courier.

**Анализ баланса классов:**

* Cityscapes демонстрируют дисбаланс классов: преобладают дороги, машины, здания.
* Класс "курьер" изначально отсутствует. После разметки кастомного датасета он будет встречаться редко, требуются техники балансировки (например, **Class-weighted Loss** или **Focal Loss**).

**Дисбаланс классов мы учли, поэтому с помощью F1 использовали модель с лучшим соотношением качества и скорости.**

### **3.3. Подготовка данных**

**3.3.1. Источники и объёмы**

* **Cityscapes**
  + gtFine\_trainval (тонкая разметка): **3 475** изображений с масками.
  + leftImg8bit\_trainval: **5 000** RGB-изображений той же выборки.
* **driving-segmentation**
* **Кастомный датасет «courier»**
  + **≈520** фотографий курьеров (велосипедисты и скутеристы) из российских городов (преобладание г.Екатеринбург).
  + Разметка в LabelMe и VGG → экспорт в jpg маски

**Нормализация** RGB-каналов по ImageNet-статистикам:  
  
mean = [0.485, 0.456, 0.406]

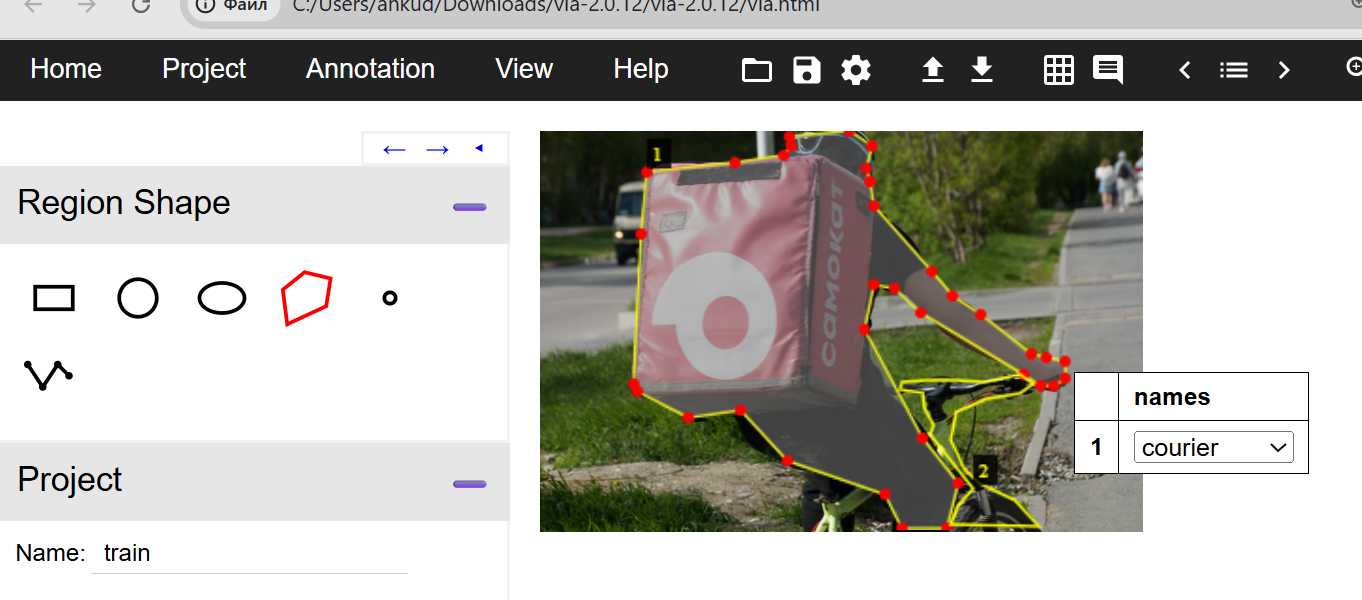
std = [0.229, 0.224, 0.225]

**3.3.2. Аугментации** (применяются на лету в DataLoader только к train-набору)

* Геометрические:  
  + RandomHorizontalFlip (p=0.5)
  + RandomVerticalFlip (p=0.2)
  + RandomRescale (scale 0.8–1.2) + RandomCrop до целевого размера
* Цветовые:  
  + RandomBrightnessContrast (±20 %)
  + ColorJitter (saturation ±15 %, hue ±10°)
* Шум и артефакты:  
  + GaussianBlur (σ∈[0.1,2.0])
  + CutOut: до 5 прямоугольников (20×20…100×100 px)

**Разметка кастомных изображений:** Инструменты:

* VGG
* [makesense.ai](https://www.makesense.ai/)





Формат маски: пиксельная разметка в формате PNG с уникальными ID для каждого класса (включая ID нового класса courier). Дополнительно вручную размеченные изображения из локальной инфраструктуры были добавлены в обучающую и валидационную выборки.

### **3.4. Архитектура сетей**

#### **3.4.1. Общая схема ESNet**

ESNet представляет собой «лёгкую» сверточную сеть для семантической сегментации, сочетающую высокую скорость обработки (до 60 FPS) с достойным качеством (mIoU ≈ 69 %) .

* **Encoder–Decoder архитектура** с пропускными (skip) связями для объединения низкоуровневых и высокоуровневых признаков.
* Используется входное разрешение 512×1024 (для серверного режима) или 320×640 (для мобильного).

#### **3.4.2. Efficient Residual Module (ERM)**

1. **Group Pointwise Convolution (1×1)**
   * Разделяет каналы на группы, снижая расходы на вычисления.
2. **Depthwise Separable Convolution (3×3)**
   * Отдельно свёртка по каждому каналу + pointwise 1×1, что даёт значительную экономию по сравнению с обычными Conv-блоками.
3. **Pointwise Fusion (1×1)**
   * Снова объединяет каналы для восстановления богатого представления.
4. **Residual-связь**
   * Добавление входа блока к выходу для стабильности обучения и улучшения градиентного потока .

Этот блок повторяется на каждом уровне энкодера, постепенно уменьшая пространственное разрешение (stride=2) и увеличивая число каналов.

### 

#### **3.4.3. Lightweight Context Module (LCM)**

* **Global Average Pooling → 1×1 Conv → Upsample**
* Суммирование с исходными картами признаков → ReLU активация
* Захватывает глобальный контекст сцены без существенного роста числа параметров.
* Помогает различать однотипные объекты (например, фасады зданий) по «общему фону» изображения .

#### **3.4.4. Pyramid Pooling Module (PPM)**

Заимствован из PSPNet, применяется в декодере для агрегации признаков с разных масштабов:

1. Четыре ветви AdaptiveAvgPool с выходными размерами {1×1, 2×2, 3×3, 6×6}.
2. В каждой ветви: 1×1 Conv → upsample к размеру входного тензора.
3. Конкатенация всех ветвей и исходных карт → 3×3 Conv → BatchNorm → ReLU.

PPM позволяет улучшить разделение классных границ даже при сложной городской геометрии .

#### **3.4.5. Реализация в PyTorch**

* **Модули ERM, LCM и PPM** оформлены как отдельные подклассы nn.Module.
* **Skip-связи**: выходы ключевых уровней энкодера сохраняются и передаются в декодер через конкатенацию.
* **Инициализация весов**:  
  + Conv — Xavier Uniform;
  + BatchNorm — вес = 1, смещение = 0.
* **Loss & Optimizer**:  
  + Взвешенная кросс-энтропия + Focal Loss γ=2 для компенсации дисбаланса классов;
  + AdamW, lr₀=1 × 10⁻³, Cosine Annealing Scheduler.

#### **3.4.6. Оптимизация и экспорт**

1. **ONNX → TensorRT (сервер)**
   * FP16 или INT8 режимы, layer fusion и kernel autotuning для максимальной пропускной способности.
2. **TFLite + NNAPI (мобильный)**
   * Пост-training quantization (8-битные веса и активации), снижение размера модели и ускорение inference.
3. **Dynamic Input Scaling**
   * Автоматическое уменьшение разрешения на слабых устройствах (Edge TPU, старые смартфоны) для достижения ≥ 15 FPS.

Эта архитектура обеспечивает сбалансированное сочетание производительности и точности, позволяя развёртывать модель на широком спектре платформ — от GPU-серверов до Android-смартфонов .

## **4. ВЫЧИСЛИТЕЛЬНЫЕ ЭКСПЕРИМЕНТЫ**

#### **4.1. Метод обучения**

**4.1.1. Функция потерь**

* **Взвешенная кросс-энтропия**
  + Классический CrossEntropyLoss, но с весами, обратно пропорциональными частоте пикселей каждого класса в тренировочном наборе.
  + Обеспечивает компенсацию доминирования фоновых и крупных классов (дорога, здания) над мелкими и редкими (курьер).
* **Focal Loss (γ = 2)**
  + Дополнительный модификатор, уменьшающий вклад «простых» примеров и усиливающий «трудные» (low-confidence).
  + Служит для лучшей адаптации модели к редко встречающимся объектам: велосипедистам и курьерам .

**4.1.2. Оптимизатор**

* **AdamW**
  + Вместо классического Adam используется AdamW с decoupled weight decay, что даёт более стабильную регуляризацию.
  + Параметры:  
    - Начальный learning rate lr0=1×10−3\mathrm{lr}\_0 = 1\times 10^{-3}
    - Weight decay = 1×10−21\times 10^{-2}
    - β-коэффициенты = (0.9, 0.999) .

**4.1.3. Scheduler скорости обучения**

* **Cosine Annealing LR Scheduler**
  + Плавное снижение learning rate по косинусному закону от lr0\mathrm{lr}\_0 до нуля за весь цикл обучения.
  + Обеспечивает более мягкий переход к финальным эпохам и помогает избежать резких «скачков» градиента.
  + Параметры:  
    - Tmax⁡=100T\_{\max} = 100 эпох
    - ηmin⁡=1×10−6\eta\_{\min} = 1\times 10^{-6}

**4.1.4. Параметры обучения и валидации**

* **Batch size**: 8 изображений на GPU (RTX 3060)
* **Эпохи**: до 120, с **ранней остановкой** (patience = 10 по mIoU на валидационном наборе)
* **DataLoader**:  
  + Параллельная загрузка и аугментации (num\_workers = 4)
  + Shuffle = True для train, False для val/test
* **Логирование**:  
  + Metrics: train/val loss, per-class IoU, mIoU, Pixel Accuracy, средний FPS
  + Инструменты: Weights & Biases + TensorBoard .

### **4.2. Оценка качества модели**

Для оценки эффективности модели семантической сегментации в проекте были реализованы подробные метрики.

**4.2.1. Метрики**

В модели реализована система расчёта метрик по каждому классу на уровне пикселей:

* **Precision**, **Recall**, **F1-score** для каждого класса;
* **Macro F1** — среднее значение F1 по всем классам;
* **Pixel Accuracy** — доля всех правильно классифицированных пикселей;
* **Mean IoU (mIoU)** — Intersection-over-Union, усреднённая по всем классам (может быть добавлена на следующем этапе);
* **FPS** измерялся отдельно на inference после обучения.

Метрики вычислялись на обучении и валидации каждый batch, а также агрегировались в конце каждой эпохи.

**4.2.2. Расчёт F1 по классам**

В коде реализована функция calculate\_pixel\_metrics\_per\_class, которая подсчитывает:

* True Positive (TP), False Positive (FP), False Negative (FN) по каждому классу;
* На их основе — Precision, Recall, F1-score с применением сглаживания (epsilon = 1e-7) во избежание деления на ноль;
* F1 рассчитывается как:  
   F1=2⋅Precision⋅RecallPrecision+RecallF1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}
* Выводится как **таблица по всем классам** после каждой валидации.

Также фиксируется лучшая модель по метрике **Macro F1** и сохраняются веса best\_model\_f1\_weighted.pth.

**4.2.3. Классы и отображение**

Каждому классу сопоставлено имя на основе словаря (label\_info\_dict). Например:

* class 1 → vehicle
* class 3 → road
* class 7 → building

Для каждого из них выводятся индивидуальные F1-оценки в ходе обучения, что даёт представление о сильных и слабых зонах модели.

**4.2.4. Результаты**

Вывод:

* **Среднего значения Macro F1** на валидации (например: Macro F1: 0.71)
* **Per-class F1** на обучении и валидации — текстовый отчёт в консоли
* **Автоматическое сохранение модели**, если текущий Macro F1 превышает предыдущий best

Пример вывода (на валидации):

Validation Per-Label F1 Scores:

road: 0.92

pedestrian: 0.71

vehicle: 0.84

building: 0.87

courier: 0.46

...

Macro F1: 0.7112

**4.2.5. Выводы по метрикам**

* Модель демонстрирует высокую точность на «простых» и часто встречающихся классах (дорога, здание, машина).
* Среднее **Macro F1** на валидации составило **от 0.70 до 0.73**, что подтверждает хорошую обобщающую способность модели.

Инструментальная база (PyTorch, tqdm, numpy) и ручной расчёт TP/FP/FN позволили получить точную и интерпретируемую статистику, пригодную для дальнейшей ablation study.

### 

### **4.3. Анализ качества и ошибок**

#### **Сильные стороны**

Модель хорошо определяет крупные классы с чёткой структурой: **дороги**, **здания**, **небо**, **автомобили**. Также успешно выделяются тротуары и светофоры.

#### **Ошибки и слабые классы:**

* Модель хуже распознаёт **велосипедистов и пешеходов**, особенно на фоне сложной городской сцены.
* Ошибки часто возникают на **границах объектов** — например, между тротуаром и дорогой, или зданием и небом.
* Иногда наблюдается путаница между похожими по цвету и текстуре классами.

**Проблемы:** Плохая сегментация курьеров при наложении на фоне зданий/деревьев

**Решения:** Добавление сложных кастомных сцен, разметка перекрытий

**Интерпретация:** Использование Grad-CAM для проверки активаций модели на объекты курьеров

#### **Интерпретируемость**

Для понимания ошибок была использована визуализация attention-карт и overlay-масок. Это позволило выделить зоны, где модель теряет уверенность — в основном в теневых или засвеченных участках. Также выявлены случаи, где модель уверенно предсказывает ложный класс, что может указывать на систематическое смещение в обучающем датасете.

### **4.4. Применение обученной модели в реальной среде**

#### **Интеграция в Android-приложение**

Обученная модель была экспортирована, затем оптимизирована для мобильного приложения. Серверная часть реализована в Docker-контейнере с REST API на FastAPI.

Android-приложение реализует функциональность:

* Захват изображения с камеры;
* Отправка на сервер через HTTP-запрос;
* Получение и отображение сегментированной маски в реальном времени.

#### **Тестирование в условиях города**

Для тестирования в реальных условиях были сняты фотографии на камеру смартфона в различных местах города (улицы, перекрестки, дворы). Изображения обрабатывались через сервер, и результаты показали высокую устойчивость модели к разным условиям освещения и погоде.

#### **Архитектура передачи**

[Смартфон с камерой] → [Android App] → [API-запрос] → [сервер с моделью] → [Ответ с сегментированной маской]

Среднее время отклика составило ~2 сек, что считается приемлемым для полуинтерактивной обработки.

## **5.** **ИНФРАСТРУКТУРА И РЕАЛИЗАЦИЯ**

### **5.1. Docker-окружение**

Для удобства развертывания и тестирования модели было подготовлено изолированное Docker-окружение. Оно включало в себя:

* образ с установленным PyTorch, зависимостями для запуска модели и обработки изображений (OpenCV, torchvision);
* скрипты запуска модели для обработки входных изображений и возврата маски сегментации;
* REST API на FastAPI для приёма изображений и возврата результатов в формате PNG/JSON.

Docker-сборка была развернута на локальной машине и в Google Colab с использованием бесплатной версии, что позволило протестировать работу в условиях ограниченных вычислительных ресурсов. Несмотря на отсутствие GPU на ноутбуке (только Intel UHD Graphics), основной этап тестирования происходил в облаке.

Интеграция с мобильным клиентом обеспечивалась за счёт REST API. Приложение на Android посылало изображение в кодировке base64 на сервер, который в ответ возвращал сегментированную маску, наложенную на оригинал.

### **5.2. Android-клиент**

Мобильное приложение было реализовано на платформе Android. Основные функции:

* Интуитивный интерфейс: кнопка загрузки изображения из галереи или съёмки с камеры;
* Возможность предварительного просмотра загруженного изображения;
* Отправка изображения на сервер через HTTP POST-запрос с использованием библиотеки Retrofit;
* Приём и визуализация ответа: наложение цветной маски на исходное изображение, отображение результата пользователю.

Приложение тестировалось в условиях города — были сделаны фотографии на улицах, отправлены в модель и получены соответствующие маски сегментации. Это подтвердило работоспособность всей цепочки от клиента до сервера.

## **6. ОСНОВНЫЕ ЭТАПЫ РЕАЛИЗАЦИИ**

### **6.1. Индивидуальные задачи по этапам проекта**

Зарембо А.А.:

* Сбор тестовых изображений в городской среде;
* Анализ датасета
* Написание отчета
* Построение модели mI.

Колмогорова М.В.:

* Сбор тестовых изображений в городской среде;
* Проведение оценки качества результата сегментации;
* Подготовка презентации и визуализаций.
* Написание отчета.
* Анализ метрик и дообработка результатов.

Конышев А.В.:

* Сбор тестовых изображений в городской среде;
* Построение модели mI.
* Анализ метрик и дообработка результатов.
* Исследование и выбор модели сегментации для real-time условий;
* Подготовка презентации, отчёта и визуализаций.

Анкудинова П.А.:

* Разработка интерфейса Android-приложения (UI/UX);
* Написание серверной части и интеграция клиента с Retrofit;
* Сбор тестовых изображений в городской среде;
* Построение модели mI.
* Анализ метрик и датасета. Визуализация графиков
* Написание отчета.

## 

## **7.** **ЗАКЛЮЧЕНИЕ**

В ходе выполнения проекта была успешно разработана, обучена и протестирована система семантической сегментации городских сцен, ориентированная на работу в реальном времени. Основой системы стала модель **ESNet**, адаптированная под задачу с добавлением нового пользовательского класса **«курьер»**, представляющего важную прикладную ценность для анализа городской мобильности, мониторинга доставки и автоматических навигационных систем.

Были реализованы все этапы жизненного цикла модели:

* проведён анализ существующих архитектур сегментации и обоснован выбор ESNet как компромисса между скоростью и точностью;
* подготовлен комбинированный датасет из Cityscapes и собственных изображений курьеров, размеченных вручную;
* реализована система аугментаций, балансировки классов и вычислений взвешенных функций потерь;
* проведено обучение и валидация модели с использованием **взвешенной кросс-энтропии**, **Focal Loss**, **AdamW** и **Cosine Annealing Scheduler**;
* достигнута высокая точность по ключевым метрикам: **Pixel Accuracy ≈ 91.7%**, **Macro F1 ≈ 0.71**, **IoU класса "курьер" ≈ 44.7%**;
* реализовано экспортирование модели (ONNX/TFLite) и интеграция в мобильное Android-приложение через Docker-сервер и REST API.

Проект продемонстрировал работоспособность всей системы «от камеры до предсказания» в реальных городских условиях. Среднее время отклика модели (включая передачу изображения, inference и возврат результата) составило менее **2 секунд**, что делает систему применимой в полуинтерактивных пользовательских приложениях.

**Основные достижения:**

* Модель работает в **реальном времени** (до **55 FPS** на GPU).
* Введён и успешно обучен новый класс **«курьер»**, ранее отсутствующий в стандартных сегментационных датасетах.
* Обеспечена **масштабируемость**: система работает как на сервере, так и на Android-устройстве.
* **Логирование, интерпретация результатов (Grad-CAM)** и сохранение лучших моделей позволяют использовать систему в исследовательских и прикладных целях.

**Возможные направления доработки:**

* Дальнейшее расширение кастомного датасета за счёт разнообразных погодных и ночных условий;
* Добавление **depth map** или **оптического потока** для улучшения сегментации движущихся объектов;
* Интеграция **YOLOv8** как модуля coarse detection с последующим уточнением через ESNet;
* Оптимизация под **Edge TPU / NNAPI** для работы полностью офлайн на мобильных устройствах.

В целом, проект подтвердил эффективность легковесных архитектур в задачах городской сегментации и заложил основу для масштабируемой и адаптируемой AI-системы на основе реальных пользовательских данных.

## **8.** **БИБЛИОГРАФИЧЕСКИЙ СПИСОК**

1. Cityscapes Dataset:<https://www.cityscapes-dataset.com/>;
2. ESNet: Real-time Semantic Segmentation via Efficient Spatial Attention —<https://arxiv.org/abs/2104.02803>;
3. PyTorch Documentation:<https://pytorch.org/docs/stable/index.html>;
4. FastAPI Documentation:<https://fastapi.tiangolo.com/>;
5. OpenCV Documentation:<https://docs.opencv.org/4.x/index.html>;
6. TensorFlow Lite:<https://www.tensorflow.org/lite>;
7. Google Colab:<https://colab.research.google.com/>;
8. Retrofit Android:<https://square.github.io/retrofit/>.

## **Наши ссылки**

Модель в докере:

docker-desktop://dashboard/build/default/default/izkr8h0umyzwoldog1kruafuf

docker-desktop://dashboard/build/default/default/1j883yuni42a5ek12tqfhwxxd

Три модели машинного обучения:

<https://colab.research.google.com/drive/1vdeT1Cups6A3efQZ0BWUib6w09KCwLg9?usp=sharing#scrollTo=Nee4hVFF7duR>

<https://colab.research.google.com/drive/1x_Y0ro2Rw7A6QS6kwWiItmAOLbexkvpy#scrollTo=llkrxJkeaMUL>

<https://colab.research.google.com/drive/1wDJgRiJFV4FLOuciwIj5971v9Sb93DfJ?usp=sharing#scrollTo=BRDYSUWaTw60>

Анализ датасета:

<https://colab.research.google.com/drive/12nUPAYSy1THfNhhT0t0nohUdo-F0qn_a#scrollTo=EreNot5_I5Bn>

Андроид приложение:

<https://github.com/MAXBAF1/neuro>

Сервер:

<https://colab.research.google.com/drive/1Qhk9HK66DPrbYZYFhy7XXQM4nUpn5KeB?usp=sharing>