МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

РАБОТА ПРОВЕРЕНА

Рецензент
Доцент кафедры ИАОУ
ФГАОУ ВО «ЮУрГУ (НИУ)», к.т.н.

_____ А.А. Шинкарев

«___»_____ 2024 г.

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Редуцирование нейронной сети для семантической сегментации изображений

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.04.04.2024.308-1501.ВКР

Научный руководитель,
доцент кафедры СП, к.ф.-м.н.
_____ Е.В. Иванова

Автор работы,
студент группы КЭ-229
_____ А.Ю. Струева

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студентке группы КЭ-229
Струевой Анастасии Юрьевне,
обучающейся по направлению
09.04.04 «Программная инженерия»
(магистерская программа «Искусственный интеллект и инженерия данных»)

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)
Редуцирование нейронной сети для семантической сегментации изображений.
2. **Срок сдачи студентом законченной работы:** 20.05.2024 г.
3. **Исходные данные к работе**
3.1. Zhou Z., Siddiquee R., Tajbakhsh N., Liang J. UNet++: A Nested U- Net Architecture for Medical Image Segmentation. // Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support, 2018. – 3–11 pp.
3.2. Iakubovskii, P. Segmentation Models Pytorch. [Электронный ресурс] URL: https://github.com/qubvel/segmentation_models.pytorch (дата обращения: 14.04.2024 г.).
4. **Перечень подлежащих разработке вопросов**
4.1. Провести обзор архитектур нейронных сетей для семантической сегментации изображений.
4.2. Провести обзор методов редуцирования нейронных сетей.
4.3. Осуществить сбор и предобработку данных для обучения нейронной сети для семантической сегментации изображений.

4.4. Выполнить реализацию нейронных сетей с использованием разных методов редуцирования.
4.5. Провести тестирование производительности работы редуцированных нейронных сетей.
5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
доцент кафедры СП, к.ф.-м.н. Е.В. Иванова

Задание принял к исполнению А.Ю. Струева

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

В настоящее время на рынке востребованы технологии с использованием нейронных сетей. Они способны адаптироваться и решать разнообразные задачи в области обработки видео, изображений, аудио и текста привлекает внимание широкого круга пользователей и бизнес-сообщества.

Использование нейронных сетей открывает перед компаниями и исследовательскими группами неограниченные возможности для создания инновационных решений, повышения производительности и качества продуктов, а также приводит к новым открытиям и достижениям в области искусственного интеллекта.

Нейронные сети имеют потенциал, превосходящий возможности классических моделей и алгоритмов машинного обучения. С их помощью мы можем достичь гораздо более высокой точности и эффективности в решении самых сложных задач. Однако этот большой потенциал сопровождается высокими требованиями к вычислительным ресурсам. С ростом точности и сложности нейронных сетей возрастает и необходимость в мощных вычислительных устройствах. Многие из современных нейронных сетей требуют наличия мощных графических процессоров для эффективной работы.

Однако не всем предоставляется возможность приобретения такого оборудования. Поэтому возникает необходимость в упрощении нейронных сетей, чтобы сделать их доступными для использования на менее мощных устройствах. Это позволит расширить круг пользователей и создать возможность для применения нейронных сетей в самых разнообразных областях, даже там, где доступ к высокопроизводительным вычислительным ресурсам ограничен [52, 55].

Постановка задачи

Целью данной работы является редуцирование нейронной сети для семантической сегментации изображения.

Для достижения поставленной цели необходимо решить следующие задачи:

1) провести обзор архитектур нейронных сетей для семантической сегментации изображений;

2) провести обзор методов редуцирования нейронных сетей;

3) осуществить сбор и предобработку данных для обучения нейронной сети для семантической сегментации изображений;

4) выполнить реализацию нейронных сетей с использованием разных методов редуцирования;

5) провести тестирование производительности работы редуцированных нейронных сетей.

Содержание работы

Работа состоит из введения, пяти глав, заключения, библиографии. Объем работы составляет 69 страниц, объем библиографии составляет 62 источника.

В первой главе описывается предметная область и проводится обзор существующих типов нейронных сетей, применяемых для задач сегментации изображений. Описаны различные архитектуры нейронных сетей, используемых в современных исследованиях и приложениях. Также рассмотрены методы редуцирования нейронных сетей, направленные на уменьшение размера сети без существенной потери ее производительности. Приведены определения метрик для оценки качества и производительности нейронных сетей. На основе проведенного обзора выбрана целевая архитектура для дальнейшего редуцирования и оптимизации. Так же были рассмотрены существующие наборы данных, и определены требования к целевому набору данных.

Во второй главе описываются функциональные и нефункциональные требования к разрабатываемой системе. На основе требований было проведено проектирование архитектуры системы для проведения экспериментов.

Приводятся диаграммы, описывающие архитектуру системы для проведения экспериментов.

В третьей главе описаны программные средства, участвующие в разработке системы, проведена настройка автоматизации системы проведения экспериментов и фиксирования результатов. Так же в данной главе проведен сбор и разметка набора данных.

В четвертой главе описываются обучений целевой нейронной сети, описывается план проведения экспериментов по редуцированию. Так же описываются проведенные эксперименты по редуцированию нейронной сети, и описания полученных результатов. Проводится анализ и сравнение полученных результатов.

В заключении представлены основные результаты выполненной работы и описаны направления, в которых возможны дальнейшие исследования.

В приложениях содержатся основные листинги реализации компонентов системы. Так же в приложении приводятся топологии базовой архитектуры нейронной сети до редуцирования и топология архитектуры нейронной сети после проведения редуцирования. В последнем приложении приводится сводная таблица с результатами проведенных экспериментов, а также примеры сегментации изображений.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Постановка задачи

Рассмотрим различные устройства и их аппаратные ограничения, которые не позволяют использовать большие архитектуры нейронных сетей. Исходя из выявленных ограничений установим требования к редуцированной модели нейронной сети, которую можно было бы использовать на устройствах с ограниченными ресурсами.

При работе модель нейронной сети потребляет определенное количество оперативной памяти компьютера и в случае параллельной обработки n-го количества изображений, данное число вырастет в n раз. К примеру, если модель нейронной сети потребляет 256 Мб оперативной памяти, то для параллельной обработки 4 изображений уже потребуется 1 Гб. Стоит отметить, что при этом мы не рассматриваем потребление оперативной памяти другими частями системы, такие как работа api, предобработка изображений, взаимодействие с базами данных и т.д.

Развертывание приложений в облачных ресурсах требует денежных расходов. Как правило, тарификация проходит за каждую единицу ресурсов (1 Гб оперативной памяти, 1 ядро CPU, 1 Гб дискового пространства). Таким образом, для масштабирования такого сервиса потребуется увеличивать лимиты для облачного сервиса, что повлечет за собой накладные расходы.

Как правило, задачи компьютерного зрения связаны с обработкой данных в реальном времени [23, 42], поэтому время обработки моделью так же немало важно. Обработка изображений на графических видеокартах позволяет ускорить процесс обработки, однако аренда серверов с GPU стоит значительно больше, чем с CPU. Также стоит учитывать ограничения графических ускорителей. Современные библиотеки для глубокого обучения, как правило, используют CUDA-ядра, доступные только на видеокартах Nvidia. Это означает, что на устройствах с другими видеокартами использовать ускорение для нейронных сетей может быть невозможно.

Время обработки изображения на графической видеокарте зависит от ее показателей, а также от размера нейронной сети. Для сверточной нейронной сети, запущенной на GPU, это время будет составлять 0,01 секунды, при этом запуск на CPU займет больше 2 секунд. Поэтому встает вопрос об ускорении нейронной сети, чтобы она так же быстро работала на CPU, как и на GPU.

Нейронные сети не только используются на компьютерах, но и встраиваются в более мелкие устройства, такие как телефоны и часы. На данных устройствах нейронные сети могут решать различные задачи, включая задачи компьютерного зрения. Например, они могут использоваться для распознавания сцен и объектов в окружающей среде, чтобы помочь пользователю найти определенный предмет или местоположение.

Стандартный телефон имеет 6–8 Гб оперативной памяти, примерно больше пятидесяти процентов занято операционной системой. В среднем на устройстве, которым пользуется человек ежедневно, свободно около 4 Гб оперативной памяти. В зависимости от типа приложения оно потребляет разное количество оперативной памяти: в среднем для почтового клиента необходимо 50–100 Мб, для приложений банков и соцсетей 200–500 Мб, игры и программы обработки фото/видео могут занимать до нескольких гигабайт оперативной памяти [27].

Умные часы обычно имеют значительно меньше оперативной памяти по сравнению со смартфонами. Например, в Apple Watch Series 6 доступно около 1 Гб оперативной памяти, в то время как Huawei Watch GT 2 Pro обладает примерно 32 Мб. Так же, как и у телефонов, часть оперативной памяти расходуется на операционную систему и другие приложения.

Таким образом, целью является получение нейронной сети, которая смогла бы запускаться на устройствах с ограниченной памятью без потребности наличия GPU на устройстве. При развертывании нейронная сеть должна иметь возможность обработать 10 изображений одновременно, в сервисе с ограничением в 512 Мб оперативной памяти, таким образом для

загрузки модели в память устройство доступно 50 Мб памяти. Нейронная сеть должна иметь возможность запуститься на смартфоне или часах, где в распоряжении примерно 15 Мб оперативной памяти. Время обработки нейронной сети должно быть менее 0,5 секунды ожидания пользователя. Для решения этой задачи необходимо провести редуцирование нейронной сети.

Редуцирование нейронной сети – это процесс оптимизации и повышения производительности нейронной сети с сохранением ее качества.

Для оценки качества сегментации часто используется метрика среднего значения интерсекции по объединению (mIoU) [60], которая показывает среднее значение степени перекрытия между предсказанными сегментированными областями и истинными масками для всех изображений.

Исходя из требований, для оценки производительности будут использоваться следующие метрики:

1) inference time – это среднее время обработки одного изображения нейронной сетью на CPU;

2) memory consumption – это количество оперативной памяти, которую занимает программа или процесс во время своей работы.

Таким образом, при проведении редуцирования нейронной сети следует стремиться к уменьшению метрик memory consumption и inference time, при этом максимально сохранить качество нейронной сети, а именно метрику mIoU.

1.2. Обзор типов сегментации

Существующие типы сегментации изображения можно условно разделить на три категории: семантическая сегментация (semantic segmentation) [3], инстанс-сегментация (instance segmentation) [54] и сегментация с помощью выделения границ объектов (boundary detection) [40].

Семантическая сегментация

При семантической сегментации каждый пиксель изображения классифицируется и относится к определенному классу объектов. Благодаря классификации каждого пикселя, семантическая сегментация обеспечивает точное выделение объектов на изображении, что может быть полезно во многих приложениях, например, в автоматическом распознавании и анализе изображений. В сравнении с детектированием объектов [42], семантическая сегментация более требовательна к вычислительным мощностям, так как требуется классификация каждого отдельного пикселя.

Инстанс-сегментация

Инстанс-сегментация различает каждый отдельный объект на изображении, даже если они принадлежат к одному и тому же классу. Таким образом, каждому объекту присваивается собственный уникальный идентификатор. Например, если изображение содержит несколько котят, инстанс-сегментация поможет отделить каждого котенка от других и присвоить им уникальные идентификаторы (рисунок 1).

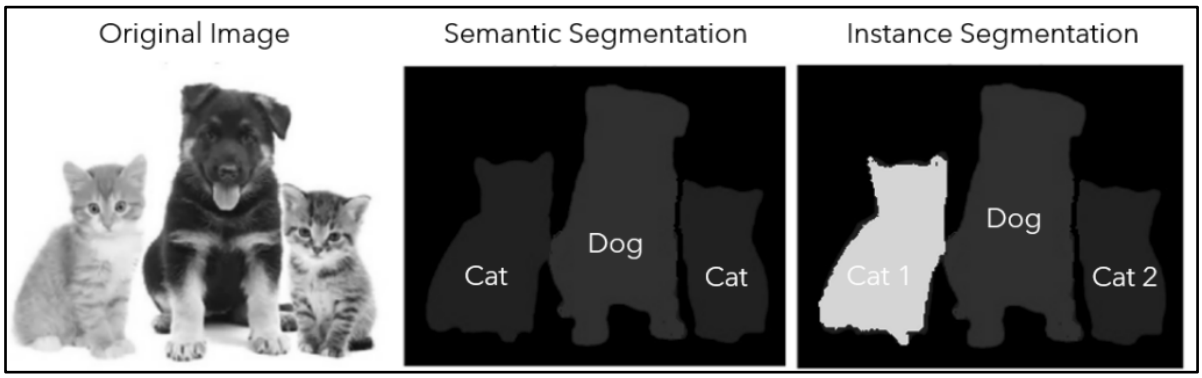

Рисунок 1 – Пример семантической сегментации и инстанс-сегментации

Один из основных плюсов инстанс-сегментации состоит в том, что она способна не только разделить изображение на семантические области, но и точно определить каждый отдельный экземпляр объекта на изображении. Выделение каждого отдельного объекта может быть востребована, к примеру, при сегментации дорожного движения. Способность точно выделять и классифицировать каждый автомобиль, пешехода, велосипедиста и

другие объекты на дороге позволяет системам безопасности принимать быстрые и точные решения, такие как предупреждение о столкновении или управление ускорителем и тормозами для предотвращения аварий. Однако инстанс-сегментаторы, в сравнении с семантическими, более требовательны к вычислительным мощностям, это делает их менее подходящими для применения в реальном времени или на устройствах с ограниченными вычислительными мощностями.

Сегментация с помощью выделения границ объектов

Выделение границ объектов может быть использоваться для сегментации изображения. Для этого на изображении сначала выделяются контуры объектов, затем происходит разделение на объекты и уже после этого все полученные замкнутые границы помечаются как отдельный сегмент (рисунок 2).

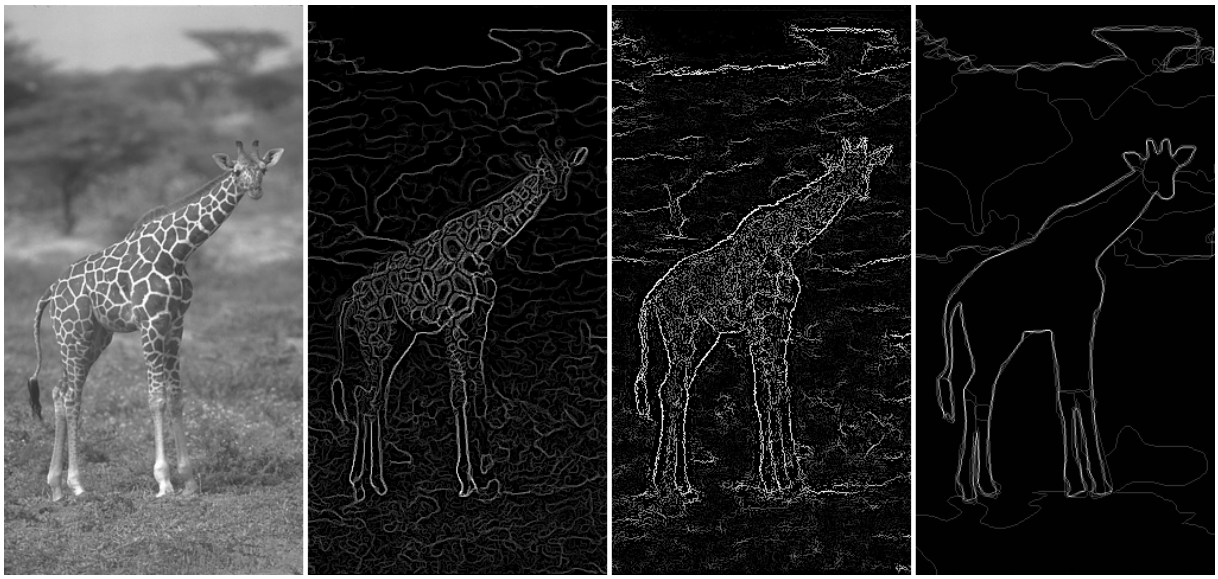

Рисунок 2 – Пример сегментации с помощью выделения границ

В основе данного подхода лежат классические алгоритмы такие как оператор Собеля [56] и алгоритм разреза на графах [12], поэтому подход очень не требует больших вычислительных мощностей и занимает меньше времени, чем методы, основанные на нейронных сетях. Главным недостатком такого подходя является то, что в некоторых сценах границы объектов

могут быть нечеткими или неоднородными из-за различных факторов, таких как освещение, тени или наложение объектов. Это может затруднить точное обнаружение границ и правильную сегментацию.

Сравнение типов сегментации изображений

Существуют два типа сегментации: семантическая сегментации и инстанс-сегментация. Данные типы решают разные задачи сегментации. Инстанс-сегментация используются там, где необходимо разделять каждый объект друг от друга, не зависимо от того относятся они к одному классу или нет. Семантическая сегментация больше подходит для анализа изображения. Задачу сегментации можно решать с помощью нейронных сетей и с помощью классических алгоритмов машинного обучения, однако нейронные сети для сегментации имеют большую точность в сравнении с классическими алгоритмами сегментации. Нейронные сети для семантической сегментации менее требовательны к вычислительным ресурсам, чем инстанс-сегментаторы, поэтому для проведения экспериментов по редуцированию нейронной сети будет использоваться семантическая сегментация.

1.3. Обзор архитектур нейронных сетей для семантической сегментации

Рассмотрим архитектуры нейронных сетей, которые осуществляют сегментацию изображений.

U-Net

Архитектура U-Net [44] представляет собой глубокую сверточную нейронную сеть, состоящую из энкодера (encoder) и декодера (decoder). Эндкодер извлекает признаки изображения на разных уровнях абстракции и уменьшает размерность пространства признаков до размерности bottleneck. Для этого часто используется метод down-sampling, который позволяет уменьшить размер изображения и объем вычислений, сохраняя при этом основные характеристики изображения. Декодер восстанавливает про-

странственное разрешение изображения, за счет чего происходит воссоздание изображения, для этого часто используется метод up-sampling, который увеличивает размер изображения, восстанавливая его детали и структуру после процесса сжатия, проведенного в энкодере. U-Net имеет механизм передачи информации из одного уровня нейронной сети в другой, такой механизм называется пропускающим соединениям (skip connections) между ними. Пропускающие соединения помогает избежать потери детализации в процессе декодирования. На рисунке 3 приведена архитектура U-Net.

U-Net++ это улучшенная версия оригинальной архитектуры [61]. Основное отличие U-Net++ от исходной архитектуры заключается в добавлении дополнительных связей между энкодером и декодером, что позволяет сети более эффективно использовать контекстную информацию и лучше справляться с проблемой потери пространственных деталей в процессе декодирования.

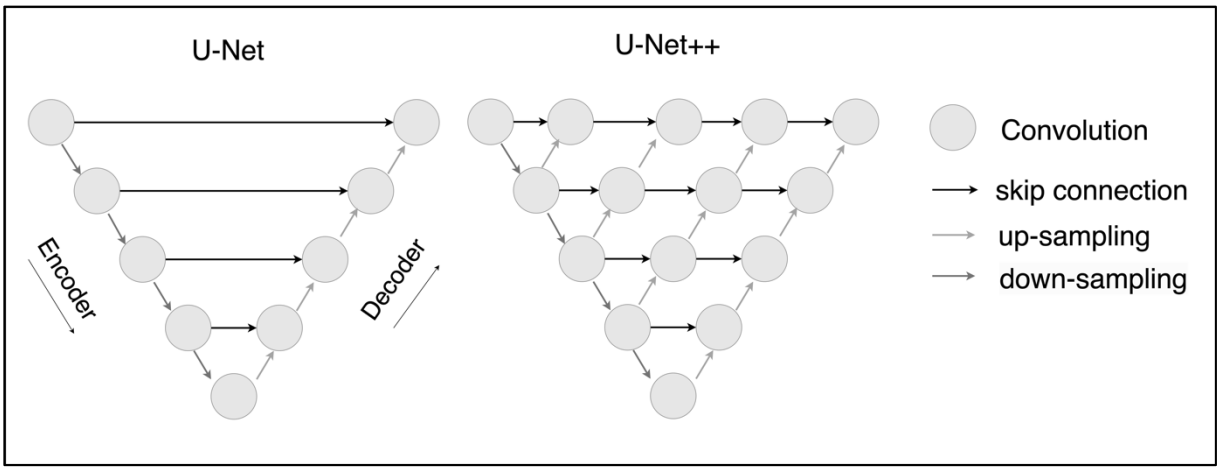

Рисунок 3 – Архитектура U-Net и U-Net++

PSPNet (Pyramid Scene Parsing Network)

Основная идея PSPNet [59] заключается в использовании пирамидальную агрегацию контекста – изображение разбивается на несколько областей с разными размерами, а затем для каждой области вычисляется статистика, отражающая ее контекст (рисунок 4). Одна из ключевых особенностей PSPNet - использование пирамидального модуля (pyramid pooling module),

который позволяет сети адаптироваться к объектам различных размеров в сцене. Этот модуль позволяет сети учитывать информацию о контексте на разных уровнях детализации, что существенно для точной сегментации объектов разного размера.

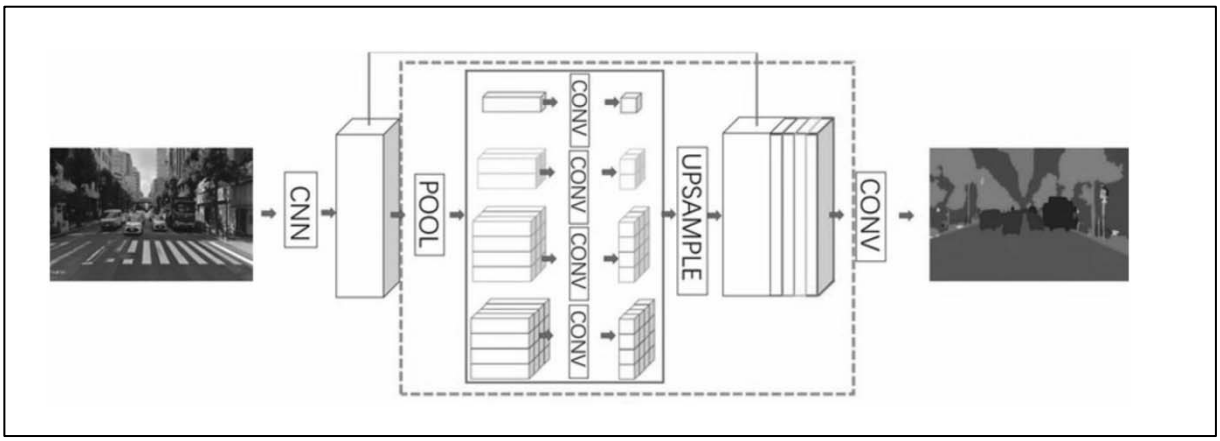

Рисунок 4 – Архитектура PSPNet

ENet (Efficient Neural Network)

ENet [23] относительно легковесная нейронная сеть, разработанная для выполнения сегментации изображений в реальном времени. Архитектура состоит из операции с небольшим количеством параметров, такие как свертки с небольшим размером ядра и шагом, что уменьшает количество вычислений, необходимых для обработки изображений.

Высокая скорость выполнения так же достигается за счет использования параллельных путей обработки для извлечения признаков на разных уровнях абстракции. Это осуществляется с помощью параллельных блоков, таких как bottleneck blocks, которые выполняются одновременно на различных частях входного изображения. Кроме того, в архитектуре ENet применяются пропускающие соединения для улучшения передачи информации между различными уровнями нейронной сети. Таким образом ENet обеспечивает хороший баланс между производительностью и точностью. На рисунке 5 изображена архитектура нейронной сети ENet.

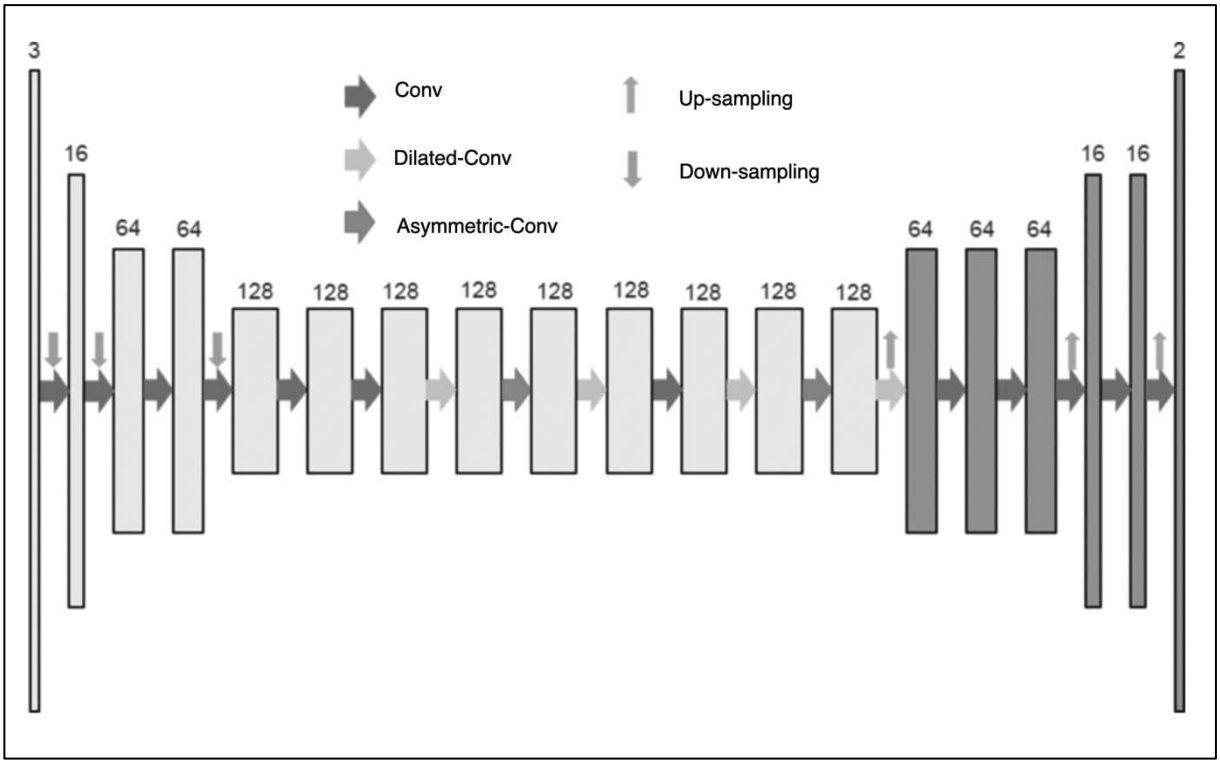

Рисунок 5 – Архитектура ENet

FCN (Fully Convolutional Network)

FCN [26] представляет собой нейронную сеть, состоящую полностью из сверточных слоев (рисунок 6). В отличии от классических сверточных нейронных сетей (convolutional neural network) для классификации изображений, FCN способна предсказывать плотные карты сегментации, где каждый пиксель изображения относится к определенному классу объектов. В отличии от ранее рассмотренных архитектур, FCN имеет прямую архитектуру и не имеет отличительных особенностей. FCN была одной из первых архитектур, специально разработанных для решения задачи семантической сегментации изображений, и ее успех способствовал появлению новых моделей и методов, которые были рассмотрены ранее.

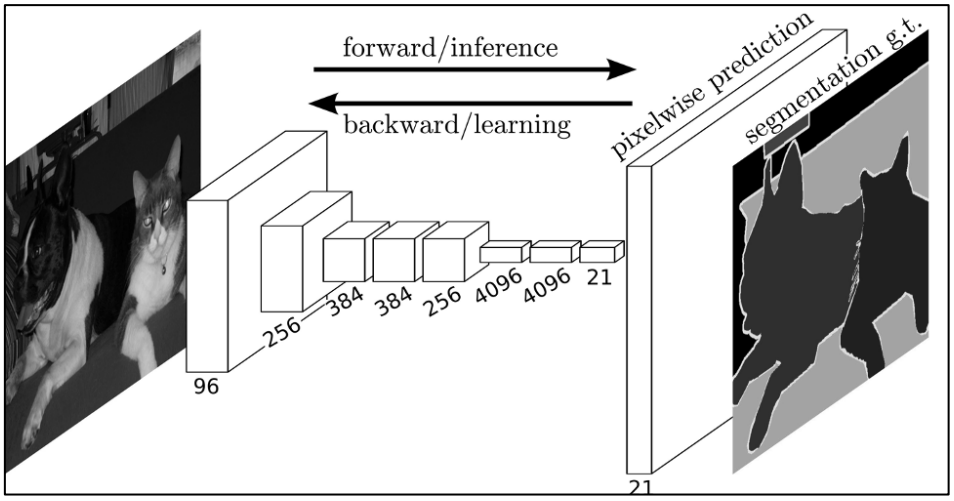

Рисунок 6 – Архитектура FCN

DeepLab

DeepLab [4] это серия из нескольких версий архитектуры, каждая из которых имеет свои особенности и улучшения. DeepLab v3+ последняя версия из этой серии и включает все улучшения предыдущих версий. Данная версия имеет энкодер-декодерную архитектуру. Так же в архитектуре применяются модифицированные атрофированные разделительные свертки (atrous separable convolutions), которые позволяют снизить количество параметров модели и улучшить ее эффективность (рисунок 7).

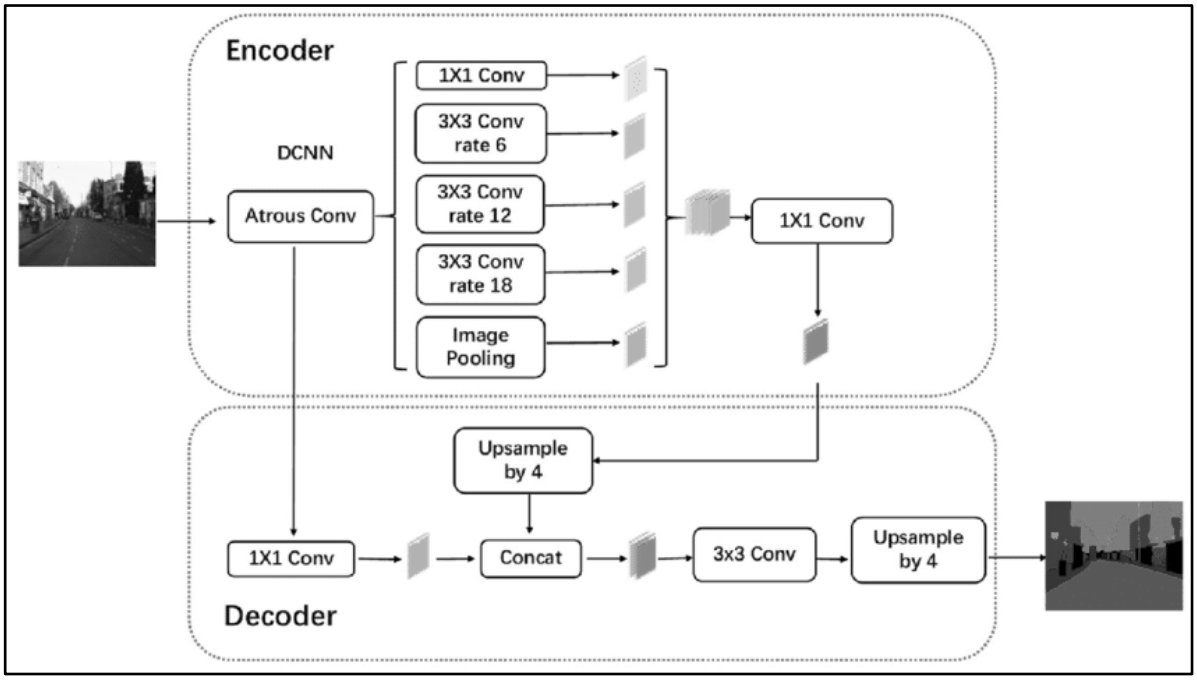

Рисунок 7 – Архитектура DeepLab

DeepLab v3+ также использует модуль асимметричного слияния (ASPP) для агрегации контекста на разных масштабах. Однако этот модуль был модифицирован и улучшен по сравнению с предыдущими версиями.

1.4. Сравнительный анализ архитектур нейронных сетей для семантической сегментации

Для выбора архитектуры нейронной сети для редуцирования был проведен анализ по метрикам качества и производительности (пункт 1.1).

Значения метрики mIoU были получены из веб-сервиса Papers with Code [6, 7]. Там предоставлены сводка показаний mIoU с указанием года исследования метрики. Данные брались из двух наборов данных: Cityscapes test и Cityscapes val, это связано с тем, что не для всех нейронных сетей предоставлены сведения о mIoU в одной таблице.

Тестирование по метрикам производительности проводились на нейронный сетях, обученных на разных наборах данных. Это связанно с тем, что исходные веса моделей не были опубликованы в интернет и нет возможности сравнить в одинаковых условиях. Однако за счет того, что архитектуры систем остаются исходные, это позволяет проводить анализ. Так же стоит отметить, что данное сравнение нацелено не на количественно сопоставлении, а скорее сопоставление подрядов этих показателей. При сравнении для всех сегментаторов инициализировался энкодер resnet101 с предобученными весами imagenet.

На рисунке 8 представлен код для измерения метрик inference time и memory consumption. Для их измерения использовался инструмент `torch.profiler`, предназначенный для профилирования производительности кода.

```
from torch.profiler import profile, record_function, ProfilerActivity

with profile(activities=[ProfilerActivity.CPU],
                         profile_memory=True,
                         record_shapes=True) as prof:
    with record_function("model_inference"):
        model(input_tensor)

inference_time = prof.key_averages()[0].cpu_time
memory_consumption = prof.key_averages()[0].self_cpu_memory_usage
```

Рисунок 8 – Измерение метрик производительности

В таблице 1 приведен анализ всех рассмотренных ранее моделей нейронных сетей, предназначенных для решения задачи семантической сегментации.

В разделе 1.1 были установлены размеры метрики, которые должна иметь итоговая редуцированная нейронная сеть. В качестве целевой нейронной сети для редуцирования будет выбираться нейронная сеть, наиболее приближенная к данным показателям.

Таблица 1 – Анализ архитектур нейронных сетей

Архитектура / **Метрика**	**U-Net++**	**PSPNet**	**ENet**	**FCN**	**DeepLabv3**
mIoU (%)[6]	–	–	58,3	65,3	81,3
mIoU (%) [7]	75,5	79,7	–	–	78,5
memory consumption (Mb)	582	101	110	478	496
inference time (сек)	4,53	0,17	0,10	2,25	2,32

Нейронные сети ENet и FCN имеют низкое качества предсказаний (метрика mIoU ниже 75%), поэтому данные нейронные сети не могут быть выбраны в качестве целевой нейронной сети.

Нейронная сеть PSPNet наиболее приближена к целевым показателям, среднее потребление памяти составляет 101 Мб, а время обработки одного изображения составляет 1,17 ссекунды. Поэтому она будет выступать в качестве целевой нейронной сети для редуцирования.

1.5. Подходы редуцирования нейронных сетей

Существующие методы редуцирования нейронных сетей можно условно разделить на четыре категории: использование более легкой архитектуры, (lightweight architectures), сжатие весов (weight compression), прореживание архитектуры (pruning), и факторизация сверточных слоев (factorized convolution) [51, 57].

Использование более легкой архитектуры

Метод использования более легких архитектур предполагает создание моделей с меньшим количеством слоев и параметров, сохраняя при этом достаточную производительность для конкретной задачи. Это может включать удаление тяжеловесных блоков из нейронной сети. Например, для задачи сегментации изображений можно заменить сложный энкодер более простым, имеющий меньшее количество слоев.

Обучение более простой модели можно выполнить двумя способами: с нуля или путем дистилляции модели. При обучении с нуля модель обучается с начальными весами, а дистилляция модели предполагает передачу знаний из более сложной модели (учителя) в более простую модель (ученика), что помогает улучшить производительность более легкой модели [50]. При обучении ученик стремится минимизации разницы между предсказаниями учителя и ученика. Существует разновидность метода дистилляции – FitNets [28], он дополнительно к передаче знаний от учителя к ученику, использует информацию о промежуточных слоях модели учителя.

Сжатие весов

Сжатие весов [22] – это уменьшения размера нейронной сети путем сокращения числа битов, используемых для хранения весов. Это позволяет уменьшить объем памяти, необходимый для хранения параметров модели, и ускорить вычисления на аппаратных устройствах.

Одним из эффективных подходов является квантизация (quantization). При квантизации все значения весов и активаций сети округляются до более

простых значений. Это позволяет уменьшить количество битов, необходимых для представления параметров модели, что снижает объем памяти, занимаемый моделью.

Предположим, модель глубокого обучения содержит 10 миллионов параметров, и каждый параметр хранится в формате float32 (4 байта). Это займет примерно 40 МБ памяти. Если мы применяем квантизацию к параметрам модели, используя 8- битные целые числа (int8), то каждый параметр будет занимать только 1 байт вместо 4 байт, что дает экономию памяти в 75%. Таким образом, модель с квантизацией будет занимать только 10 МБ, что значительно меньше, чем в исходной модели.

Прореживание архитектуры

Прореживание архитектуры [53] это метод редукции нейронных сетей, заключается в удалении лишних параметров модели, таких как нейроны, связи между нейронами и даже целые слои, с целью уменьшения размера сети и повышения ее эффективности (рисунок 9).

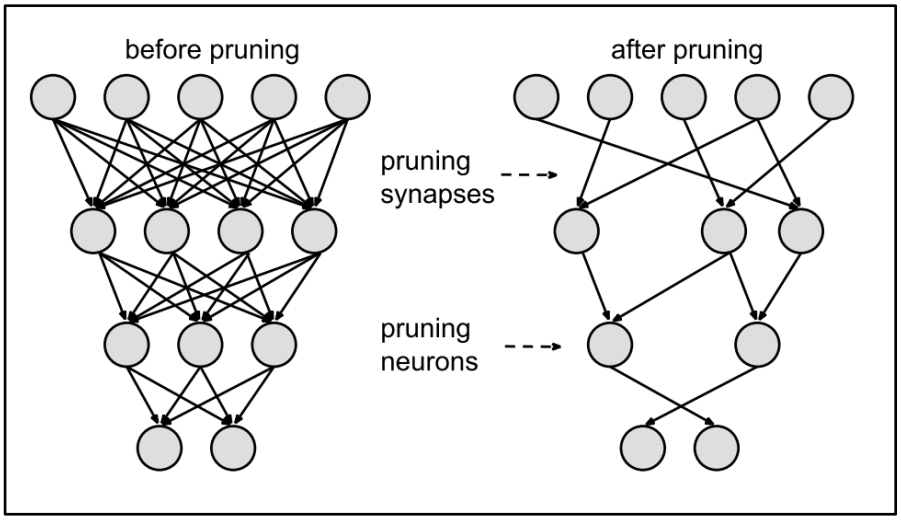

Рисунок 9 – Прореживание нейронных сетей

Все алгоритмы прореживания можно разделить на два типа: прореживание на уровне модели и эфемерное прореживание. В первом случаи веса из сети устраняются окончательно, для каждого примера из обучающей или тестовой выборке используются те же веса при инференсе (inference). Во

втором случае веса из сети не убираются, но при каждом проходе в вычислении функции используется только часть параметров модели или активаций с прошлых слоев. В качестве известных примеров можно привести dropout, где часть случайно выбранных весов инициализируется нулями [35].

Прореживание нейронной сети может осуществляться тремя разными способами: прореживание размерности после обучения (one-shot-pruning), итерационное прореживание (iterative pruning) и прореживание перед обучением (pruning before training). На рисунке 10 продемонстрированы данных подходы.

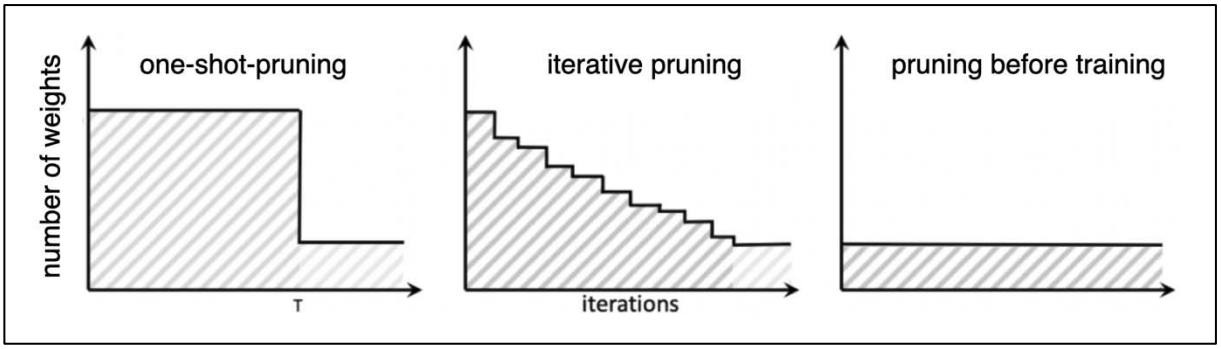

Рисунок 10 – Подходы к прореживанию нейронных сетей

Для прореживания нейронных сетей на уровне модели применяются различные алгоритмы, которые определяют, какие параметры можно удалить без значительной потери качества модели.

Одним из таких подходов является магнитудное прореживание (magnitude-based pruning). Этот метод основан на идее того, что небольшие веса или фильтры в нейронных сетях имеют меньшее значение и вносят меньший вклад в общую работу сети. Следовательно, их можно удалить без существенной потери качества модели. Принцип работы метода состоит в следующем: сначала нейронная сеть обучается на тренировочном наборе данных, затем веса или фильтры значение которых ниже порога, удаляются из сети.

Главным недостатком магнитутного прореживания является то, что некоторые веса могут иметь низкую абсолютную величину и быть удаленными, но при они могли быть важными работы сети. Эта проблема решается в методе прореживание на основе разложения Тейлора (Taylor expansion-based pruning). В данном метода удаление параметров происходит на основе анализа какие параметры вносят наименьший вклад в общую функцию потерь. Параметры с наименьшим влиянием могут быть удалены из нейронной сети без существенного влияния на ее производительность.

В процессе обучения нейронной сети также можно настроить ограничения диапазона значений весов, одним из таких подходов является L2 Regularization-based Pruning. В процессе обучения нейронной сети к функции потерь добавляется слагаемое, которое представляет собой штраф за большие значения весов. Это слагаемое выражается квадратичной нормой весов и учитывается в общей функции потерь модели.

L2 Regularization-based Pruning имеет аналогичную проблему, как и магнитудное прореживание: удаляя или обнуляя веса с меньшим вкладом в общую функцию потерь, мы можем потерять важные паттерны или закономерности в данных, которые могли быть захвачены этими весами. Эту проблему решает прореживание на основе градиентов (gradient-based pruning). Этот метод учитывает, как веса влияют на общую функцию потерь в процессе градиентного спуска.

Факторизация сверточных слоев

Факторизация сверточных слоев – это техника оптимизации для улучшения эффективности сверточных слоев в нейронных сетях. Вместо применения стандартной свертки с большим количеством фильтров, факторизация сверток разбивает сверточный слой на последовательность более простых операций, что может уменьшить количество параметров и вычислительную сложность.

Можно выделить три основных подхода к факторизации: глубинно-раздельная свертка (depthwise separable convolution), групповая свертка (group convolution), разреженная свертка (dilated convolution).

Глубинно-раздельная свертка это две более легкие операции свертки: свертка по каналам (depthwise convoluton) и свертка по точкам (pointwise convolution). Свертка по каналам – это тип свертки, в котором используется ядро 1x1: ядро, которое выполняет итерацию по каждой отдельной точке [33]. Свертка по глубине – это тип свертки, при котором применяется один сверточный фильтр для каждого входного канала [9]. В обычной свертке, выполняемой по нескольким входным каналам, фильтр имеет такую же глубину, как входной, и позволяет нам свободно смешивать каналы для генерации каждого элемента на выходе.

Для упрощения нейронной сети можно заменить сверточные слои в нейронной сети на более легкие и эффективные свертки, такие как свертка по глубине (depthwise convolution) и свертка по каналам (pointwise convolution) [20]. Эти слои представляют собой более легкие и эффективные альтернативы классическим сверточным слоям и позволяют сократить количество параметров и ускорить вычисления.

Групповая свертка подразумевает разделение слоя на группы, и каждая группа выполняет операцию свертки независимо. Это уменьшает количество связей между входными и выходными каналами и позволяет снизить сложность модели [16].

При использовании разреженной свертки происходит увеличение размер ядра свертки с помощью шага, называемого параметром dilatation или dilation rate (рисунок 11). Это позволяет увеличить размер поля зрения без увеличения количества параметров.

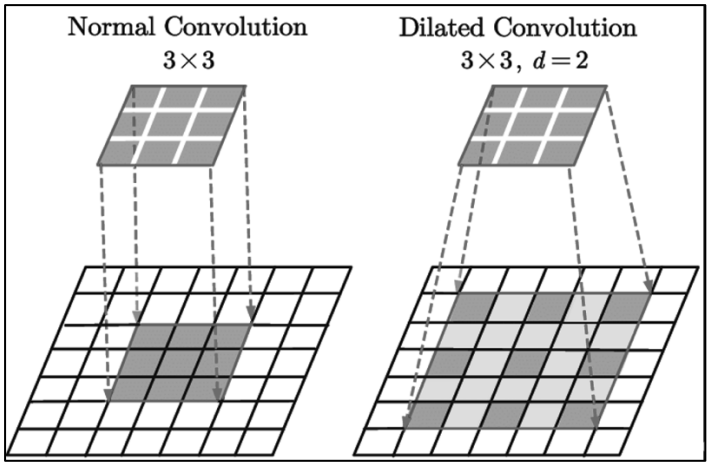

Рисунок 11 – Прореженная свертка

1.6. Обзор наборов данных для семантической сегментации

Редуцирование нейронной сети требует итеративного обучения нейронной сети, поэтому каждая итерация обучения не может требовать много времени. Исходя из этого, набор данных для обучения нейронной сети должен быть небольшим и иметь маленькое количество целевых классов для сегментации.

Существует множество наборов данных для семантической сегментации [1, 5, 8, 31]. В таблице 2 приведены названия наиболее популярных наборов данных для семантической сегментации, количество изображений, а также количество целевых классов для сегментации.

Таблица 2 – Анализ наборов данных

	COCO [8]	**Cityscapes** [5]	**ADE20K** [1]	**Pascal Voc** [31]
Количество данных	200 000	5000	20 000	10 000
Количество классов	80	30	150	20

Количество данных зависит от количества классов. В таких наборах данных как COCO и ADE20K количество классов достигает до 100, следовательно, данных там тоже больше. В двух других наборах данных классов меньше, соответственно и данных так же меньше.

Для того, чтобы ускорить процесс проведения экспериментом, можно выбрать набор данных с наименьшим объемом. Из рассмотренных наборов

данных наиболее маленьким является cityscapes. Однако он имеет 30 целевых классов, что может повысить сложность обучения нейронной сети. Поэтому для проведения экспериментов по редуцированию нейронной сети будет подготовлен собственный набор данных с небольшим количеством классов и небольшим объемом данных. Это позволит быстрее и эффективнее оценить влияние редуцирования на качество модели и ее производительность.

В качестве целевого набора данных для обучения будет подготовлен набор данных для сегментации рекламы на фасадах зданий. Анализ рекламы на изображениях фасадов зданий может помочь урбанистам и градостроителям понять, как реклама влияет на облик и эстетику городской среды. Это позволяет разрабатывать более эффективные стратегии размещения и регулирования рекламных конструкций в городе.

Рекламные элементы, такие как вывески, баннеры и плакаты, создают визуальный шум и загромождение, что затрудняет восприятие окружающей среды и снижает качество ее визуальной обстановки. Так же это может привести к утрате эстетической привлекательности городской среды и нарушить ее гармонию и целостность.

С точки зрения урбанистики можно выделить следующие типы загрязнений: граффити, вывески, баннеры, плакаты, кондиционеры и вентиляционные системы. Сегментация будут выделять два класса – фон и рекламное зашумление.

Всего было выделено 2 класса для сегментации, что примерно в 15 раз меньше, чем у набора данных cityscapes, поэтому количество данных для целевого набора данных так же можно уменьшить примерно в 15 раз. Таким образом набор данных для сегментации рекламы должен состоять приблизительно из 400 изображений.

Выводы по первой главе

В этой главе был проведен обзор предметной области. Были выявлены требования для итоговой редуцированной нейронной сети: нейронная сеть

должна иметь возможность запуска на устройствах без GPU, при обработке одного изображения потреблять менее 50 Мб оперативной памяти, время обработки изображения должно быть менее 1 секунды.

Для достижения нужных размеров были рассмотрены методы редуцирования нейронных сетей. Для оценки качества редуцирования были определены метрики качества и производительности. Для достижения хороших результатов часто применяют комбинацию методов редуцирования нейронных сетей [12, 51, 57]. После сокращения размера сети может снизиться ее точность, поэтому может потребоваться последующее дополнительное обучение нейронной сети.

Все рассмотренные подходы редуцирования начинают с исходной нейронной сети, которую затем оптимизируют. Для выявления исходной нейронной сети были рассмотрены некоторые из существующих архитектур для задачи сегментации и проведен их сравнительных анализ по установленным метрикам. Среди рассмотренных архитектур можно выделить PSPNet в качестве идеального баланса между потреблением ресурсов и скоростью работы.

Для того, чтобы проведение экспериментов проводилось быстро, будет собран и размечен собственный простой набор данных для сегментации рекламного шума на фасадах зданий. Набор данных будет включать приблизительно 400 изображений с 2 классами.

2. ПРОЕКТИРОВАНИЕ

2.1. Определение требований

Для проведения экспериментов и взаимодействия с моделью нейронной сети был спроектирована система для автоматизации проведения экспериментов. В ходе проектирования были определены функциональные и нефункциональные требования к разрабатываемой системе.

Функциональные требования

Функциональные требования определяют функциональность программного обеспечения, то есть описывают, какое поведение должна предоставлять разрабатываемая система.

Разрабатываемое приложение должно удовлетворять следующим функциональным требованиям:

1) нейронная сеть должна выполнять семантическую сегментацию изображения;

2) проведение экспериментов должно быть автоматизировано;

3) система должна предоставлять хранилище для набора данных для обучения;

4) система должна проводить подсчет метрик качества и производительности нейронной сети;

5) система должна предоставлять возможность просмотра результатов обучения.

Нефункциональные требования

К нефункциональным требованиям системы относятся свойства, которыми она должна обладать. Например, удобство использования, безопасность, расширяемость и т.д.

Система должна удовлетворять следующим нефункциональным требованиям, которые описаны ниже.

1. Система должна предоставлять API на языке Python 3 [38] для взаимодействия с нейронной сетью.

2. Проведение экспериментов и развертывание API должно быть автоматизировано с помощью механизма CI/CD [14].

3. Система должна использовать библиотеку pytorch [39] для работы с нейронной сетью.

4. Эксперименты должны проводиться с использованием инструмента для управления версиями данных и моделей машинного обучения DVC [10].

5. Нейронная сеть должна быть менее требовательна к вычислительным ресурсам в сравнении с базовой нейронной сетью, при этом точность нейронной сети не должна быть ниже, чем на 5% от исходной.

2.2. Диаграмма вариантов использования системы

В соответствии с требованиями была построена диаграмма вариантов использования системы, которая представлена на рисунке 12. При проектировании был использован язык графического описания для объектного моделирования UML [62]. Диаграмма отражает взаимодействие действующих лиц (actors) «ML-инженер» и «Пользователь» с системой. Действующее лицо – это пользователь, который взаимодействует с системой.

Действующему лицу «Пользователь» доступен вариант использования API «Загрузить изображение для сегментации». После загрузки изображения сервис загрузит его и начнет процесс сегментации изображения. Так же пользователю доступен вариант использования «Получить результат сегментации». После загрузки изображения запускается процесс сегментации, проводятся необходимые преобразования и выполняется вызов модели. После чего, готовая сегментированная маска возвращается обратно пользователю системы.

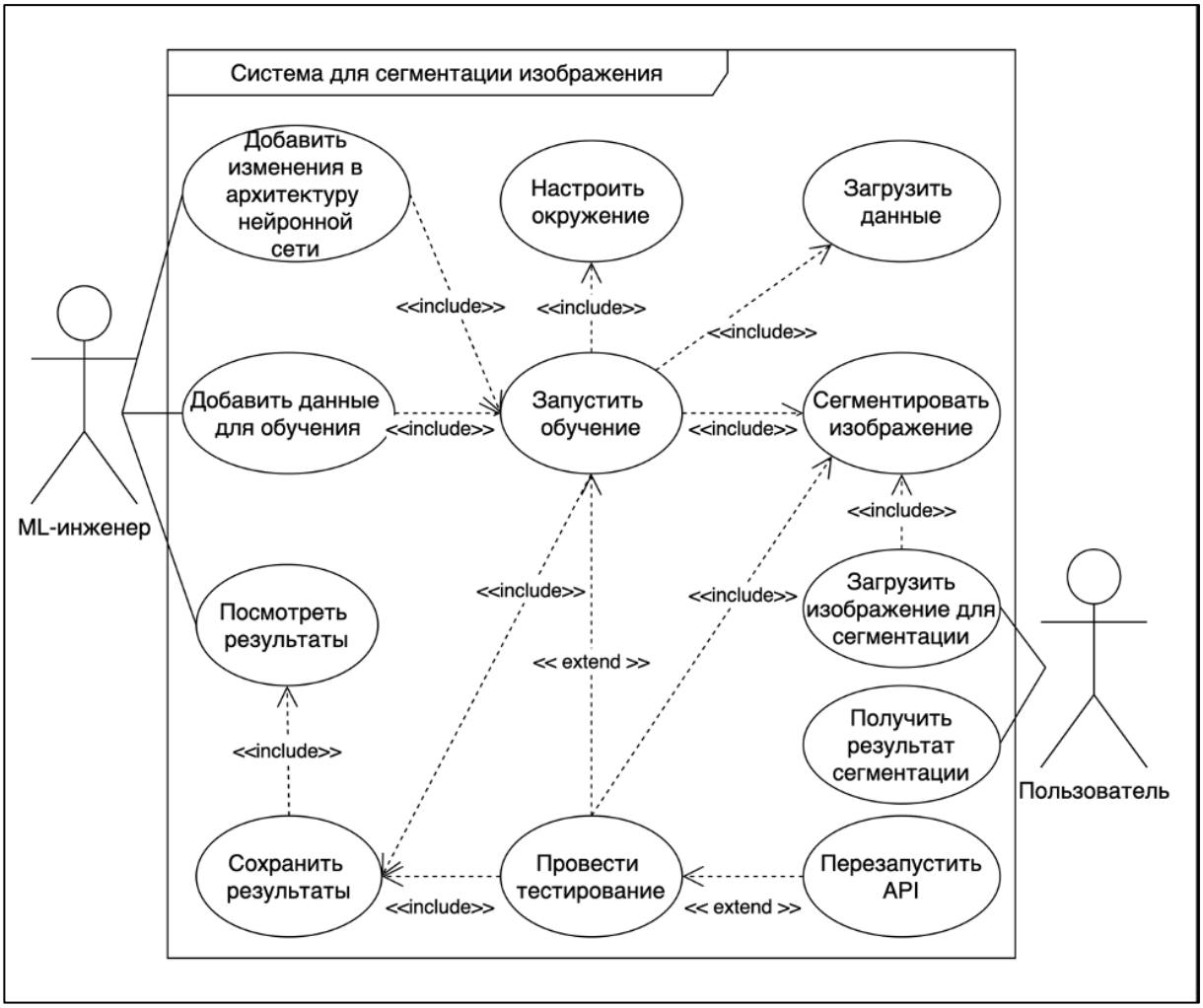

Рисунок 12 – Диаграмма вариантов использования

Действующее лицо «ML-инженер» взаимодействует с системой как разработчик модели машинного обучения и ему доступны следующие варианты использования.

1. Вариант использования «Добавить изменения в архитектуру нейронной сети». Пользователь может добавить изменения в архитектуру нейронной сети. После чего начнется обучения нейронной сети. Перед началом обучения система настроит окружение для обучения, загрузит необходимые данные. В процессе обучения нейронной сети будет итеративно приводиться сегментация изображения, а также сохранения результатов обучения. При завершении обучения будет произведено тестирование нейронной сети и в случае, если метрики улучшились произойдет перезапуск API с новой моделью нейронной сети.

2. Вариант использования «Добавить данные для обучения». Пользователь может добавить данные для обучения, при этом аналогично варианту использования «Добавить изменения в архитектуру нейронной сети» запустится обучение нейронной сети.

3. Вариант использования «Посмотреть результаты». Пользователь может посмотреть результаты обучения и тестирования нейронной сети.

2.3. Архитектура системы

В соответствии с выявленными требованиями и вариантами использования системы было проведено проектирование архитектуры. Для наглядного представления архитектуры системы была построена диаграмма размещения (рисунок 13).

ML-инженер производит у себя на персональном компьютере сбор и подготовку обучающих данных, разработку новых преобразований функций, разработку и редуцирование архитектуры нейронной сети, а также настройку различных гиперпараметров для ее обучения. После завершения он делает отправку изменений в Git репозиторий (push to remote repository). Затем механизм CI/CD запускает поочередно этапы разработки приложения: build, experimentation, и deploy.

На этапе build производится сборка docker-образа с нужными для работы зависимостями. При создании образа он сохраняется в Registry, для того чтобы его можно было использовать в дальнейшем. Все другие этапы клонируют данный образ для своей работы. Запуск данного этапа производится только при изменении файла конфигурации dockerfile или при добавлении новых данных. На этапе test производится запуск тестирования функций системы.

На этапе experimentation происходит скачивание набора данных для обучения из сервиса для разметки данных Roboflow [43] и происходит предобработка данных и их разбиение на обучающую, вариационную и тесто-

вую выборки. Затем начинается обучения нейронной сети. На каждой итерации обучения происходит вызов модели и подсчет метрик точности. После обучения происходит тестирование системы, где изменяются метрики качества и производительности нейронной сети, описанные в разделе 1.1. Все полученные результаты тестирования фиксируются с помощью DVC. Результаты тестирования затем доступны в веб-интерфейсе Iterative Studio.

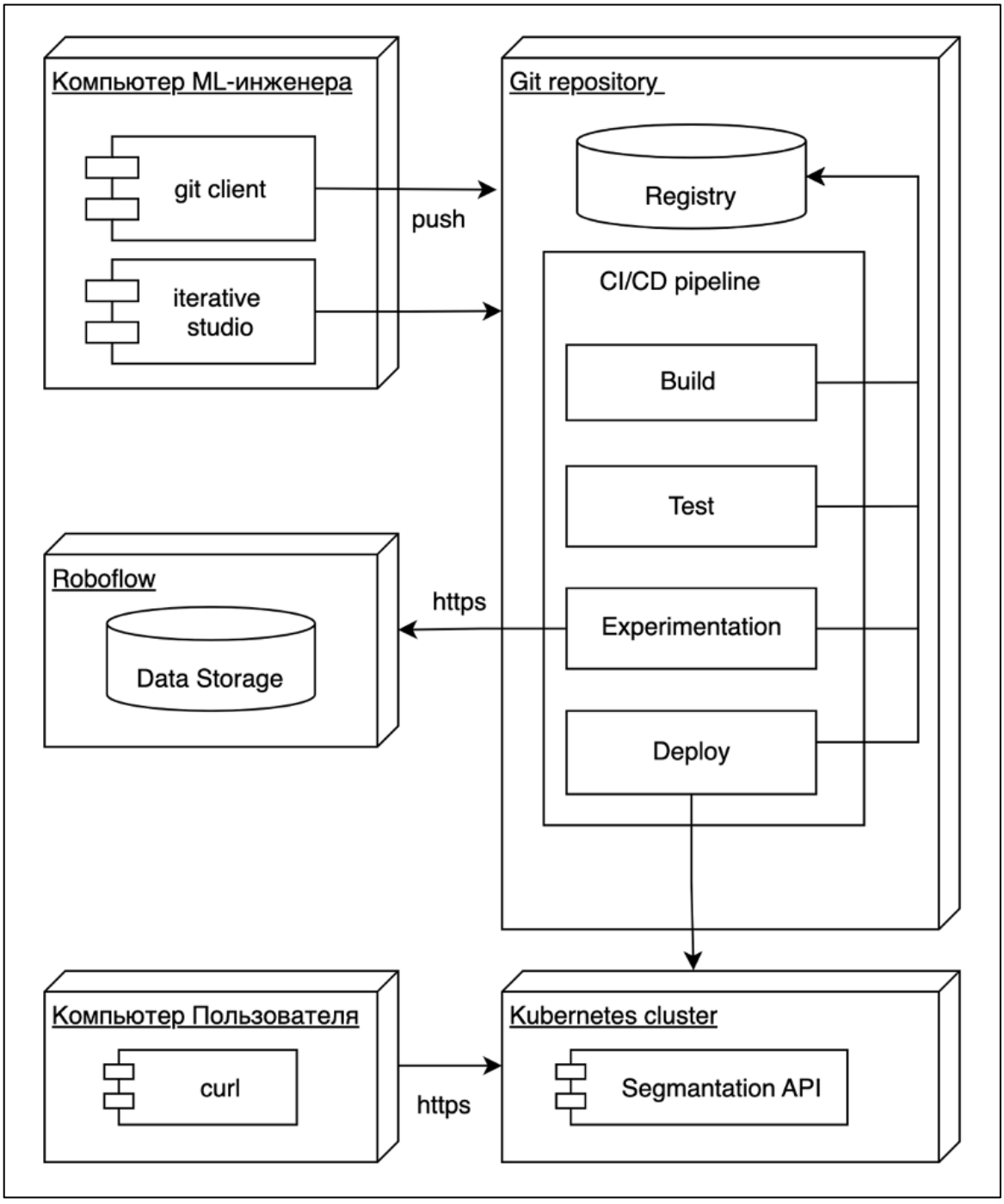

Рисунок 13 – Диаграмма размещения

На последнем этапе deploy происходит проверка метрик модели, в случае если модель улучшила свои показания, происходит автоматическая регистрация сервиса segmentation API в кластере kubernetes [25].

Segmentation API будет имеет единственную конечную точку. Для определения соглашений для взаимодействия с сервисом был написан контракт (рисунок 14). Контракт описывает формат запроса, используемый методы и входные параметры. При вызове конечной точки будет происходить загрузка изображения, вызов модели для сегментации и возвращение пользователю сегментированного изображения.

```
{
  "endpoint": "/segmentator-api",
  "method": "POST",
  "request_type": "multipart/form-data",
  "request_fields": {
    "image": {
      "description": "Изображение для сегментации.",
      "format": "form-data"
    }
  }
}
```

Рисунок 14 – Контракт для конечной точки Segmentation API

Выводы по второй главе

Во второй главе были определены функциональные и нефункциональные требования для системы автоматизации проведения экспериментов. Была спроектирована архитектура системы для проведения экспериментов и рассмотрены ее основные компоненты.

Работа с набором данных и редуцирование нейронной сети будет проходить локально на компьютере разработчика. Обучение, тестирование и развертывание модели должно быть автоматизировано на основе механизма CI/CD. Все полученные эксперименты будут фиксироваться с помощью DVC и будут доступны в веб-интерфейсе Iterative Studio.

3. РЕАЛИЗАЦИЯ

Программные средства реализации

Для разработки программной части приложения был выбран высокоуровневый язык программирования Python версии 3.10.9. Разработка велась в редакторе исходного кода Visual Studio Code внутри контейнеров разработки (devcontainers). Контейнеры разработки представляют изолированную среду, содержащую все необходимое для работы, включая зависимости, инструменты и настройки.

Серверная часть приложения была реализована с использованием библиотеки Flask версии 3.0.3 [13].

Для работы с нейронными сетями были использованы следующие библиотеки: PyTorch версии 2.2.2 [39], torchvision версии 0.17.2 [39], pillow версии 10.3.0 [32], numpy версии 1.19.5 [29], albumentations версии 1.4.3 [2], scikit-learn версии 1.4.2 [48], pandas версии 2.2.2 [30].

Для тестирования приложения использовалась библиотека pytest версии 8.1.1 [36].

Для настройки всех зависимостей использовалась библиотека poetry версии 1.2.2 [37].

Для работы с DVC использовалась библиотека версии 3.49.0 [10]. DVC это инструмент, который создан для управления версиями моделей и данных в ML-проектах.

Для автоматизации проведения экспериментов и запуска этапов обучения и тестирования нейронной сети использовался механизм GitLab pipeline.

3.1. Настройка CI/CD

Для настройки GitLab pipeline необходимо настроить конфигурационный файл, в который определяются все этапы проведения экспериментов. В листинге 1 приложения А приведен листинг настройки файла gitlab-ci.yml. Были настроены все четыре этапа проведения экспериментов (рисунок 15).

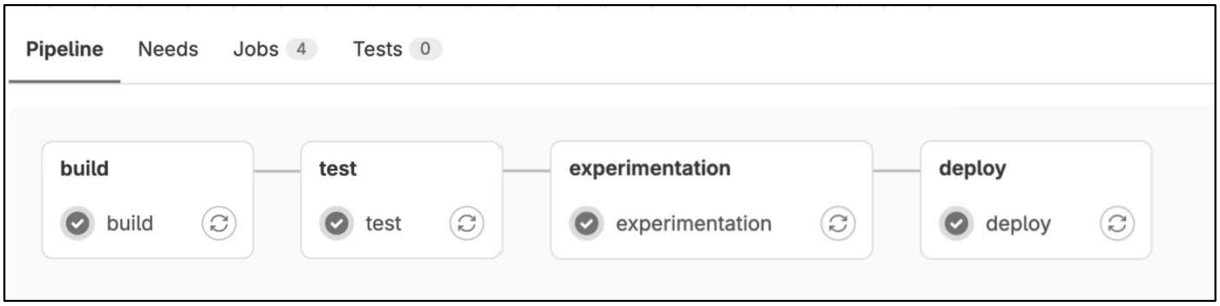

Рисунок 15 – GitLab pipeline

На этапе build происходит сборка docker-образа для дальнейшей работы. Для настройки окружения был создан dockerfile (приложение А, листинг 2), в котором определены все необходимые шаги для настройки среды разработки, а также установка библиотек, требуемые для обучения нейронной сети и запуска API. Для управления зависимостями используется инструмент poetry, который автоматизирует процесс установки и управления зависимостями проекта. В качестве базового образа использовался образ python:3.10.9-slim.

Для ускорения обучения и работы все этапы запускались на ранере (runner) с видеокартой. Runner – это дополнительные узлы, которые обеспечивают выполнение задач и автоматическое выполнение процессов CI/CD. Использование ранеров с видеокартой позволяет ускорить обучение нейронной сети за счет параллельной обработки и использования вычислительных возможностей GPU. Обучение проходило на видеокарте NVIDIA Tesla A10 [34] со следующими характеристиками: 24 ГБ видеопамяти GDDR6, количество ядер до 10240 ядер CUDA, частота ядра 1,3–1,5 ГГц.

Для хранения файлов размером выше 100 Мб был настроен Git LFS. Для этого он заменяет эти файлы на указатели, сохраняя сами файлы в отдельном хранилище, что позволяет ускорить операции коммита, ветвления и слияния.

На этапе test происводится тестирование системы с помощью библиотеки pytest. На этапе deploy осуществляется отправка запроса в

кластер на разворачивание севиса segmantation API с нужными ему переменными среды.

Регистрируемый сервис представляет собой REST API и содержит реализацию метода, описанного в пункте проектирования. Для реализации был использован фреймворк Flask. Рисунок 16 отображает реализацию функции, которая вызывается при обращении к конечной точке `/segmantetion`.

В методе `segmantetion` происходит загрузка файла, преобразование полученного файла, вызов модели для сегментации. В случае успешного выполнения выше действия, осуществляется отправка изображения пользователю, иначе выводится сообщение об ошибке.

```
model = FasadSegmentation()

@model_blueprint.route('/', methods=['POST'])
def upload_file():
    image = request.files['image']

    image_bytes = image.read()
    mask = model.predict(image_bytes)
    mask = io.BytesIO(mask.numpy().tobytes())
    return send_file(mask, mimetype='image/jpeg')
```

Рисунок 16 – Реализация конечной точки segmentation

3.2. Реализация функций для обучения нейронной сети

На рисунке 17 отображен цикл обучения нейронной сети. На каждой итерации цикла из обучающего загрузчика данных `train_loader` извлекается пара изображение-маска. Затем модель принимает изображение в качестве входных данных и выдает прогнозы. После этого вычисляется значение функции потерь между прогнозами и маской. Для оценки производительности модели также вычисляются метрики mIoU и точность пикселей. Далее происходит обратное распространение ошибки, оптимизатор обновляет веса модели с помощью метода `optimizer.step()`, а затем градиенты обнуляются `optimizer.zero_grad()`. Также в процессе обучения использовался планировщик скорости обучения `lr_scheduler`, который изменяет

скорость обучения в течение каждого этапа обучения с увеличением и уменьшением скорости обучения для более эффективного обучения сети. По мере выполнения цикла суммируются значения функции потерь и метрик производительности для последующей оценки.

```
for i, data in enumerate(tqdm(train_loader)):
    image, mask = data
    output = model(image)
    loss = criterion(output, mask)
    train_iou_score += mIoU(output, mask)
    train_accuracy += pixel_accuracy(output, mask)
    loss.backward()
    optimizer.step() #update weight
    optimizer.zero_grad() #reset gradient
    scheduler.step()
    train loss += loss.item()
```

Рисунок 17 – Цикл обучения одной эпохи

Для запуска цикла обучения и отслеживания всех экспериментов использовался DVC. Для его настройки нужно задать конфигурацию файла params.yaml (приложение А листинг 3). Все переменные параметры были вынесены в конфигурационный файл dvc, для удобства настраивать и задавать все параметры.

На рисунке 18 приведен пример фиксирования результатов тестирования в DVC.

```
with Live() as live:
    live.log_metric(f'loss', loss, plot=True, timestamp=True)
    live.log_params(metrics)
```

Рисунок 18 – Фиксирование результатов в DVC

На рисунке 19 приведен пример отображение результатов в Iterative Studio. Iterative Studio предоставляет различные инструменты для визуализации данных, включая графики, диаграммы, тепловые карты и т. д. Кроме того, оно может включать функционал для сравнения различных моделей, анализа производительности и сопоставления результатов с ожидаемыми показателями.

Plots 2 Trends Compare

Created ↓	Message	CML	dvclive/params.yaml			
					test	
			inference_time	memory_used	mIoU	train_time
Apr 24, 2024	feat: train without bathnormalisation		0.06	33	0.855309453...	3207.850748...
Apr 24, 2024	feat: train with 15 output stride		0.06	36	0.855737943...	3284.821541...
Apr 23, 2024	feat: train with 1 in_channel		0.05	35	0.779229484...	2032.008721...
Apr 23, 2024	feat: train with 3 channel		0.09	35	0.847512119...	3237.217411...
Apr 22, 2024	feat: train PSPNet ResNet 18		0.06	35	0.852386794...	3457.012897...
Apr 22, 2024	feat: train PSPNet ResNet 34		0.07	43	0.857695944...	3515.833452...
Apr 22, 2024	feat: train PSPNet ResNet 50		0.17	102	0.861367335...	25630.26558...
Apr 21, 2024	Merge remote-tracking branch 'origin/main' i...		0.19	102	0.848407638...	105779.0890...
Apr 21, 2024	feat: train PSPNet ResNet 101		0.19	102	0.848407638...	105779.0890...

Рисунок 19 – Отображение результатов в Iterative Studio

3.3. Подготовка набора данных

Для обучения нейронной сети было подготовлено 400 фотографий фасадов зданий города Челябинск. На рисунке 20 представлен пример собранных изображений.

Рисунок 20 – Пример размеченных данных

Разметка данных проводилась с помощью сервиса Roboflow [43]. Данный сервис предоставляет инструменты для разметки и предобработки изображений. На рисунке 21 представлен пример процесса разметки изображения в сервисе Roboflow. Данный сервис так же предоставляет возможность хранение и версионирование наборов данных. Подготовленные наборы данных можно получить по сгенерированной сервисом ссылке для скачивания.

Рисунок 21 – Интерфейс Roboflow

Для загрузки данных был создан класс DataPreparer, который загружает данные из хранилища и разделяет их на три выборки: обучающую, валидационную и тестовую. Во время обучения нейронной сети используется DataLoader, который итерируется по набору данных. Для каждого загружаемого изображения применяется аугментация – процесс увеличения разнообразия тренировочных данных путем изменения изображений. Аугментация изображений осуществляется с использованием библиотеки albumentations. Рисунок 22 отражает функции преобразований для обучающей и тестовой выборок.

```
train_transforms = A.Compose([A.Resize(704, 1056,
                                       interpolation=cv2.INTER_NEAREST),
                              A.HorizontalFlip(),
                              A.VerticalFlip(),
                              A.GridDistortion(p=0.2),
                              A.RandomBrightnessCon-
trast((0,0.5),(0,0.5)),
                              A.GaussNoise()])

val_transforms = A.Compose([A.Resize(704, 1056,
                                       interpolation=cv2.INTER_NEAREST),
                            A.HorizontalFlip(),
                            A.GridDistortion(p=0.2)])

test_transforms = A.Resize(768, 1152, interpolation=cv2.INTER_NEAREST)
```

Рисунок 22 – Функции преобразования для наборов данных

Выводы по третьей главе

Во третьей главе была проведена разработка системы для проведения экспериментов по редуцированию нейронной сети. Был настроен GitLab CI/CD для автоматизации всех этапов жизненного цикла нейронной сети. Так же был реализован сервис для взаимодействия с нейронной сетью, который доступен в публичном доступе.

Была рассмотрена разработка модулей для предобработки данных и модуля для обучений нейронной сети. Также была проведена настройка сохранения результатов в сервис DVC и осуществлен сбор набора данных. Итоговый набор данных содержит 400 изображений фасадов зданий с рекламой города Челябинск.

4. ПРОВЕДЕНИЕ ЭКСПЕРИМЕНТОВ

4.1. Обучение базовой нейронной сети

В качестве базовой архитектуры для редуцирования была выбрана архитектура PSPNet. В качестве энкодера используется ResNet101, который обеспечивает высокую точность при извлечении признаков из изображений разного содержания. Декодер состоит из четырех операций пулинга с размерами 1x1, 2x2, 3x3 и 6x6, что позволяет модели агрегировать информацию с различных масштабов. На выходе декодера находится модуль классификации, который определяет принадлежность каждого пикселя к определенному классу объекта.

Модули для построения нейронной сети были взяты из библиотеки `segmentation_models.pytorch`. Для инициализации весов были взяты предобученные веса модели ResNet101 на наборе данных ImageNet. Для инициализации сверточных слоев в декодере используется метод Kaiming, слои batch-normalization инициализируются таким образом, чтобы среднее значение было близко к 1, а смещение к 0, для полносвязанных слоев используется метод Xavier. На рисунке 23 приведен код для инициализации весов декодера.

```
import torch.nn as nn
for m in module.modules():

    if isinstance(m, nn.Conv2d):
        nn.init.kaiming_uniform_(m.weight, mode="fan_in", nonlinear-
ity="relu")

    elif isinstance(m, nn.BatchNorm2d):
        nn.init.constant_(m.weight, 1)
        nn.init.constant_(m.bias, 0)

    elif isinstance(m, nn.Linear):
        nn.init.xavier_uniform_(m.weight)
```

Рисунок 23 – Инициализация слоев

В процессе обучения нейронной сети были проведены эксперименты по подбору гиперпараметров, наилучшие результаты были достигнуты с помощью следующей комбинации: в качестве функции потерь применялась

перекрестная энтропия, оптимизатором был выбран AdamW с коэффициентом скорости обучения 0,0001. Для планирования скорости обучения использовался OneCycleLR. Размер пакета (batch size) составлял 3 изображения. Обучение длилось в течение 15 эпох, среднее время обучения одной эпохи составило 2 часа.

Метрика mIoU нейронной сети на тестовой выборке составила 84,8%, метрика inference time составила 0,19 секунд, memory consumption 102 Мб. На рисунке 24 отображены графики метрики mIoU и ошибки loss на валидационной выборке в процессе обучения. На рисунке 3 приложения В приведены примеры сегментации изображений.

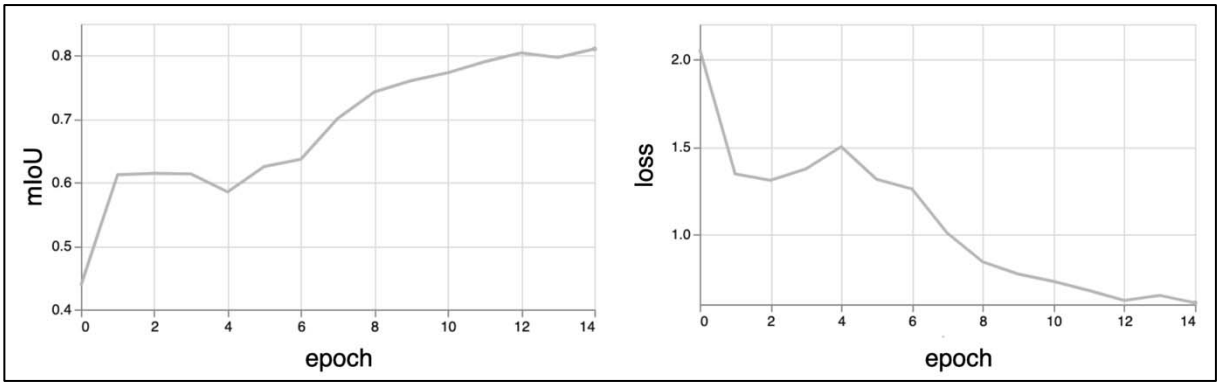

Рисунок 24 – Графики обучения нейронной сети

4.2. План проведения экспериментов

Первым этапом редуцирования нейронной сети (рисунок 25) будет получение более легкой архитектуры нейронной сети и обучение данной архитектуры с нуля. Использование именно этого метода позволит повысить производительность модели на первых этапах, а также значительно упростить модель, что в дальнейшем уменьшит время ее обучения, что позволит проводить эксперименты более быстро. Обучение нейронной сети будет проводиться с нуля, так как оно будет значительно быстрее чем дистилляция модели. В случае, если точность модели сильно снизится, определенный метод редуцирования можно будет осуществить в последнюю очередь и обучить модель с помощью дистилляции.

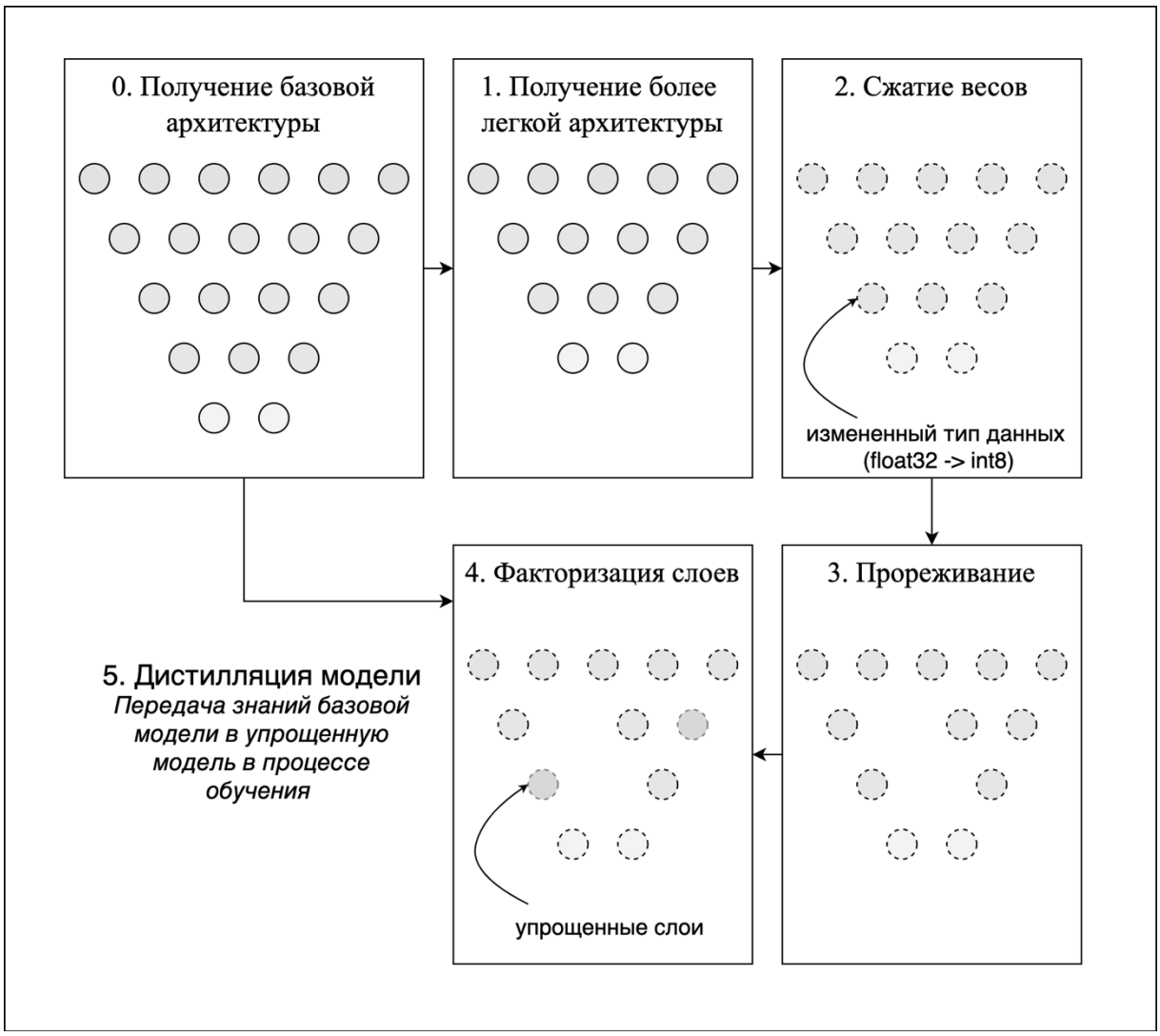

Рисунок 25 – Этапы редуцирования нейронной сети

На втором этапе будет проводиться сжатие весов, поскольку данный метод может существенно уменьшить размер модели. Применение этого метода позволяет уменьшить объем хранения весов примерно в 4 раза, что является значительным приростом производительности.

Прореживание нейронной сети будет осуществляться на третьем этапе, так как этот метод подразумевает удаление отдельных нейронов в нейронной сети. Данный метод может дать меньший прирост по сравнению с предыдущими подходами.

Четвертым этапом будет факторизация нейронной сети. Это вызвано тем, что в процессе обучения нейронная сеть инициализируется с использо-

ванием предварительно обученных весов. Поэтому, если в ходе факторизации происходит изменение структуры слоев, это может привести к тому, что веса, которые уже были настроены на определенные признаки в предыдущей модели, могут потерять свою значимость или стать неприменимыми.

Последним этапом будет дистилляция модели. В случае обнаружения методов редуцирования нейронной сети, которые могут значительно улучшить производительность модели, но при этом снижают ее точность, можно провести дистилляцию.

4.3. Использование более легкой архитектуры

Наиболее тяжелой частью данной архитектуры является энкодер, который составляет примерно 70% от размера всей нейронной сети. Замена энкодера на более простой, имеющий меньшее количество слоев в архитектуре, позволяет значительно снизить размер нейронной сети и уменьшить время ее обучения. В ходе экспериментов были использованы энкодеры ResNet с различным количеством слоев: 50, 34 и 18 (таблица 3).

При уменьшении размера энкодера точность модели осталась на уровне исходной. При этом объем потребляемой памяти и время выполнения модели сократились примерно в три раза. Время обучения также уменьшилось с 29 часов до одного часа. Дальнейшие эксперименты будут проводиться относительно модели с энкодером ResNet18.

Таблица 3 – Изменение размера энкодера

Метод редуцирования	mIoU (%)	inference time (сек)	memory consumption (Мб)	train time (сек)
ResNet101	84,8	0,19	102	105780
ResNet50	86,1	0,17	102	25630
ResNet34	85,7	0,07	43	3515
ResNet18	85,2	0,06	35	3460

Было выполнено прореживание на уровне сети: снижение количества входных и выходных каналов, удаление слоев batch-normalization из энкодера и декодера и уменьшение глубины энкодера (таблица 4).

Уменьшение количества входных каналов с 3 до 1 не привело к значительному улучшению производительности модели, но при этом точность нейронной сети значительно снизилась. Это объясняется тем, что при объединении трех RGB каналов в один GRAY канал теряется важная информация о цветовой составляющей изображения, которая является важной для задачи сегментации.

Таблица 4 – Изменение параметров нейронной сети на основе ResNet18

Метод редуцирования	**mIoU (%)**	**inference time (сек)**	**memory consumption (Мб)**	**train time (сек)**
ResNet18	85,2	0,06	35	3460
Снижение количества входных слоев до 1	77,8	0,05	35	2032
Удаление слоев batch-normalization из декодера	85,5	0,06	33	3207
Снижение размера выходного слоя до 256	85,3	0,04	33	3560
Снижение размера выходного слоя до 128	84,5	0,06	33	3582
Снижение глубины энкодера до 1	83,3	0,06	26	3012
Снижение глубины энкодера до 2	78,4	0,04	22	3367
Удаление слоев batch-normalization из энкодера	82,7	0,06	22	3697

Удаление слоев batch-normalization помогло снизить количество потребляемой памяти, при этом точность нейронной сети не снизилась. Это произошло потому, что batch-normalization в основном используется для ускорения обучения путем стабилизации распределения активаций на каждом слое в процессе обучения. Во время использования модели нет необходимости в этой стабилизации, так как модель уже обучена, и активации должны быть стабильными без дополнительного выравнивания. Однако даже если слои batch-normalization отключаются во время работы модели, они все равно присутствуют в структуре нейронной сети и занимают память. Таким образом, отключение batch-normalization помогает снизить потребление памяти и ускорить обработки изображения моделью, даже если они все еще присутствуют в структуре модели.

В изначальной версии модели размер выходного слоя составлял 512 нейронов. В процессе экспериментов было проведено уменьшение размера выходного слоя до 256 и 128 нейронов. При уменьшении до 128 точность модели снизилась на 1%, при этом производительность незначительно увеличилась. Поэтому для дальнейших экспериментов будет использоваться модель с размеров выходного слоя декодера 256.

Уменьшение глубины энкодера подразумевает отключение некоторых слоев нейронной сети. Несмотря на увеличение производительности, точность модели значительно снизилась. Поэтому эта оптимизация требует применения дистилляции модели, так как обучение с нуля в данном случае не подходит.

В качестве лучшей модели будет взята модель с 3мя входными каналами, и размером выходного слоя 256 и отключенными слоями batch-normalization в энкодере и декодере.

4.4. Сжатие весов

Одним из вариантов сжатия весов является использование более простого типа данных для построения нейронной сети. Исходный тип данных в модели float32, использование целочисленного типа данных помогло бы уменьшить модель в 4 раза. Однако для построения модели использовалась библиотека pytorch, которая работает с тензорами. Использование целых чисел в качестве типа данных для тензоров приведет к недифференцируемости операций, что делает их непригодными для применения в алгоритмах оптимизации, таких как градиентный спуск.

Pytorch поддерживает два типа вещественных числе: float16 и bfloat16. Их отличие в том, что float16 использует 1 бит для знака, 5 бит для экспоненты и 10 бит для мантиссы, в то время как bfloat16 использует 1 бит для знака, 8 бит для мантиссы и 7 бит для экспоненты. Тип bfloat16 был специ-

ально разработан для применения в глубоком обучении и обеспечивает баланс между точностью и эффективностью, что делает его более подходящим для широкого спектра задач машинного обучения.

При использовании типа данных float16 было необходимо преобразовывать данные в формат float32 перед некоторыми операциями интерполяции, поскольку эти операции не поддерживают float16. Потребление памяти при этом сильно снизилось и составило 17 Мб, однако время выполнения модели в таком случае выросло до 3,56 секунд (таблица 5).

Таблица 5 – Сжатие весов

Метод редуцирования	**mIoU (%)**	**inference time (сек)**	**memory consumption (Мб)**	**train time (сек)**
Удаление слоев batch-normalization из энкодера	82,7	0,06	22	3697
Использование типа данных float16	–	3,56	17	–
Использование типа данных bfloat16	–	2,85	12	–
Использование квантования после обучения	44,8	0,04	11,65	3650

Был проведен эксперимент с использованием типа данных в bfloat16. В данном случае перевод данных в float32 не потребовался, так как все слои поддерживают тип данных bfloat16. Потребление памяти при этом сильно снизилось и составило 12 Мб, однако время выполнения модели в таком случае выросло до 2,85 секунд. Так происходит, потому что в некоторых случаях, операции с bfloat16 могут требовать дополнительных преобразований или обработки, что приводит к увеличению времени выполнения. Кроме того, операции с float32 могут выполняться более эффективно, поскольку так же оптимизированы и на уровне аппаратного обеспечения.

Другим методов сжатия весов является квантизация чисел. В PyTorch есть поддержка трех типов квантизации с помощью квантовых модулей

torch.nn.quantized: динамическая квантизация, (dynamic quantization), статическое квантизация после обучения (post-training static quantization) и квантование в процессе обучения (quantization aware training) [35].

Наилучшим для данных типов моделей является квантование после обучения [21], поэтому применялось именно оно. Квантизация после обучения применялась ко всем доступным для квантизации слоям, в текущей архитектуре это слои conv2 и relu. Для того чтобы использовать данный метод квантизации необходимо добавить два слоя в прямой проход модели и после обучения использовать следующие функции: `torch.quantization.QuantStub()` и `torch.quantization.DeQuantStub()`. На рисунке 26 приведен код для применения квантизации для данной модели.

```
modules_to_fuse = [['encoder.conv1', 'encoder.relu'], . . .]
model = torch.quantization.fuse_modules(model, modules_to_fuse)

model.qconfig = torch.quantization.get_default_qconfig('fbgemm')
torch.quantization.prepare(model, inplace=True)
torch.quantization.convert(model, inplace=True)
```

Рисунок 26 – Квантование параметров нейронной сети

После проведения сжатия весов, точность модели значительно снизилась, что делает данный подход непрактичным для такой небольшой нейронной сети. Учитывая незначительный размер сети, снижение точности нецелесообразно. Таким образом, преобразование весов становится невозможным для данного типа архитектуры нейронной сети.

4.5. Прореживание нейронной сети

Для прореживания нейронных сетей библиотека torch предоставляет внутренний метод `torch.nn.utils.prune`. На рисунке 27 представлен пример использования функции прореживания.

```
for name, module in model.named_modules():
     if isinstance(module, torch.nn.Conv2d):
        prune.l1_structured(module, name='weight', amount=0.2)
```

Рисунок 27 – Прореживание нейронной сети

Прореживание проводилось после обучения после обучения нейронной сети, так как для этого подхода не требуется проведения дополнительного обучения. Для прореживания были применены четыре различных функции: l1_unstructured, ln_structured, global_unstructured и l2_unstructured.

Каждая из функций имеет параметр amout, который показывает процент нейронов, которые необходимо удалить. На рисунке 28 показана зависимость потребления памяти от увеличения параметра amount.

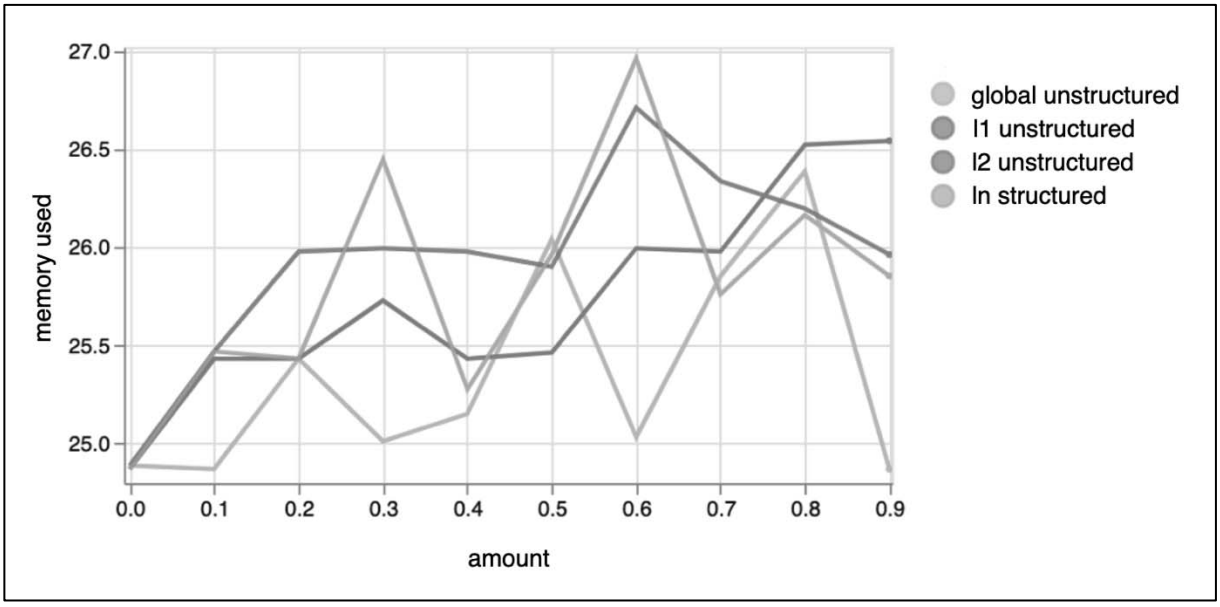

Рисунок 28 – Потребление памяти при увеличении процента удаленных нейронов для различных функций прореживания

С увеличением процента удаленных нейронов, потребление памяти выросло. Это связано с тем, что при прореживании происходит замена имени параметра его сокращенной версией, в то время как исходный (необрезанный) параметр сохраняется в новом параметре с именем `name+'_orig'` [15]. Поэтому в процессе прореживания модель нейронной сети может немного увеличиваться. Из-за того, что модели изначально маленькая, процент прореженных весов стал менее заметен, чем добавление дополнительных параметров в веса.

Было проведено эфемерное прореживание с помощью dropout. На рисунке 29 показана зависимость потребления памяти от увеличения параметра dropout. Как видно из рисунка, при увеличении процента инициализации нулями нейронов потребление памяти снижается незначительно.

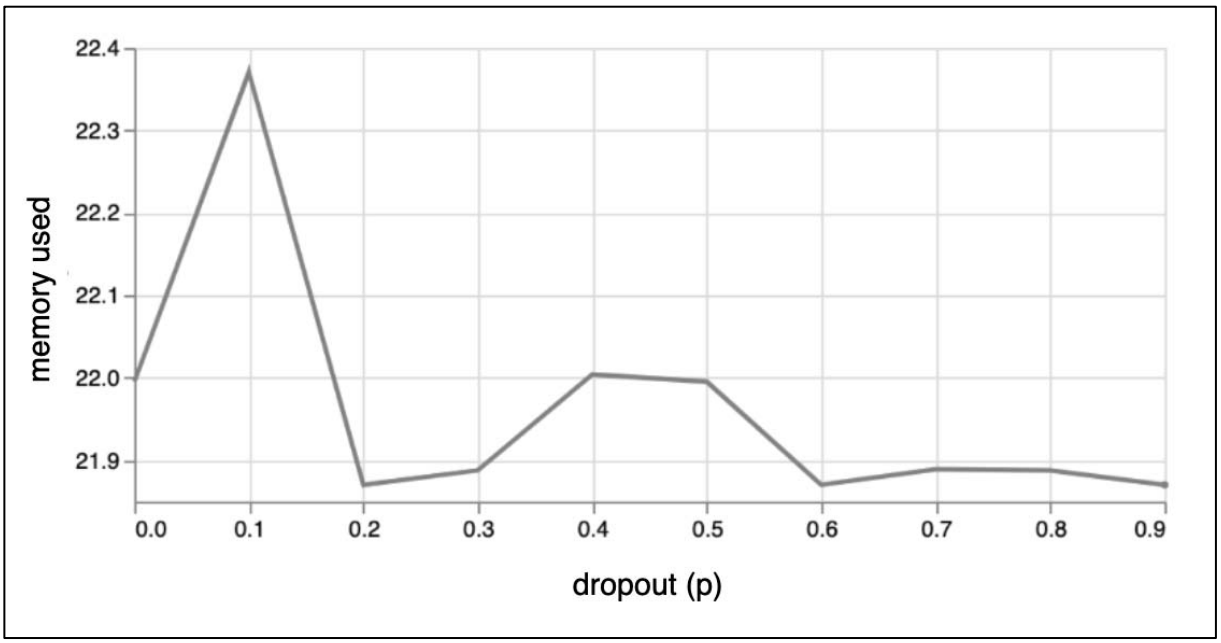

Рисунок 29 – Потребление памяти при увеличении процента удаленных нейронов для эфемерного прореживая с помощью dropout

Таким образом, прореживание нейронной сети любыми методами не имеет смысла. В случае использования стандартных методов прореживания для данной архитектуры наблюдается увеличение потребления памяти, а при использовании эффективного прореживания с помощью dropout уменьшение потребления памяти оказывается незначительным.

4.6. Факторизация сверточных слоев

Была проведена факторизация различными методами для различных сверточных слоев в архитектуре. На рисунке 30 приведен пример факторизации сверточных слоев тремя различными методами.

```
import torch.nn as nn
base_conv = nn.Conv2d(in_channels, out_channels,
                      kernel_size=7, stride=2, padding=3)
depthwise_separable_conv = nn.Sequential(
            nn.Conv2d(in_channels, in_channels,
                      kernel_size=7, stride=2,
                      padding=3, groups=3),
            nn.Conv2d(in_channels, channels,
                      kernel_size=1, stride=1, padding=0),)
group_conv = nn.Conv2d(in_channels, out_channels,
                       kernel_size=7, stride=2, padding=3, groups=3)
dilated_conv = nn.Conv2d(in_channels, out_channels,
                         kernel_size=7, stride=4, padding=3)
```

Рисунок 30 – Факторизация сверточных слоев

В таблице 6 отображена матрица, в которой указано, какой метод применялся для каждого слоя и какой прирост по производительности он дал. В данном случае не проводилось обучение нейронной сети, чтобы сократить время проведения экспериментов.

Таблица 6 – Факторизация сверточных слоев

Название слоя	Depthwise Separable Convolution	Group Convolution	Dilated Convolution
encoder.conv1	–	–	+
encoder.basicbloak.conv1	–	+	–
encoder.basicbloak.conv2	–	+	–
encoder.downsample.conv1	–	–	–
decoder.psp.poll.conv1	–	–	–
encoder.downsample.conv1	–	–	–
segmentation_head.conv1	+	–	–

Из таблицы видно, что многие методы не привели к улучшению оптимизации. Это объясняется тем, что многие сверточные слои уже были оптимизированы и представляли собой тип pointwise convolution.

Было проведено обучение нейронной сети с факторизованными слоями. На рисунке 31 отображен график метрики mIoU на валидационной выборке в процессе обучения.

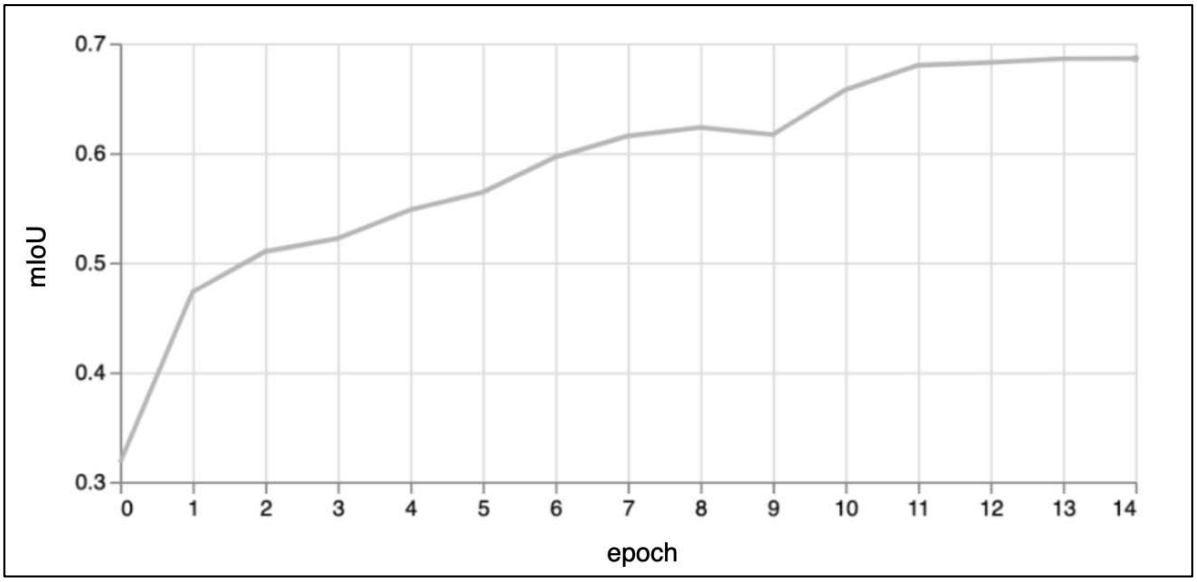

Рисунок 31 – Обучение нейронной сети с факторизованными слоями

Метрика mIoU после 11 эпох обучения перестала увеличиваться, это значит, что нейронная сеть вышла на плато. Точность на тестовой выборке составила 73,7%. Данное снижение критично, так как оно больше 5%. Поэтому данный метод следует проводить с помощью дистилляции модели.

4.7. Дистилляция модели

Эксперимент с факторизацией сверточных слоев показал значительное улучшение производительности, но при этом метрика mIoU сильно снизилась. Для того, чтобы повысить точность нейронной сети можно проводить обучение не с нуля, а с помощь дистилляции.

На рисунке 32 приведен код для дистилляции модели. Для проведения дистилляции модели, сначала происходит проход вперед с моделью учителя, а затем осуществляется прямой проход с моделью студента [24]. При этом градиенты для модели учителя не сохраняются, так как веса учителя не изменяются.

```
for _, data in enumerate(tqdm(train_loader)):
    image, mask = data
    with torch.no_grad():
        teacher_output = teacher_model(image)

    output = student_model(image)

    soft_targets = nn.functional.softmax(teacher_output / T, dim=-1)
    soft_prob = nn.functional.log_softmax(output / T, dim=-1)

    soft_targets_loss = torch.sum(soft_targets * (soft_targets.log() -
soft_prob)) / soft_prob.size()[0] * (T**2)
    label_loss = criterion(output, mask)

    loss = 0.25 * soft_targets_loss + 0.75 * label_loss
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()

    scheduler.step()
```

Рисунок 32 – Дистилляция модели

Для смягчения предсказаний учащихся применяется операция `softmax`, а затем `log()`. После этого рассчитываются ошибка, которая масштабируется по T^2, где T – температура дистилляции [50]. Итоговая потеря представляет собой взвешенную сумму двух видов потерь: потери мягких целей и жесткие потери.

Обучение нейронной сети длилось в течение 100 эпох, среднее время обучения одной эпохи составило 3 минуты. Метрика mIoU нейронной сети на тестовой выборке составила 84,3%, метрика inference time составила 0,03 секунд, memory consumption 4,5 Мб.

На рисунке 33 отображены графики метрики mIoU на валидационной выборке в процессе обучения. Из графика видно, что после 80 эпохи обучение точность вышла на плато, это значит, что обучение нейронной сети можно останавливать.

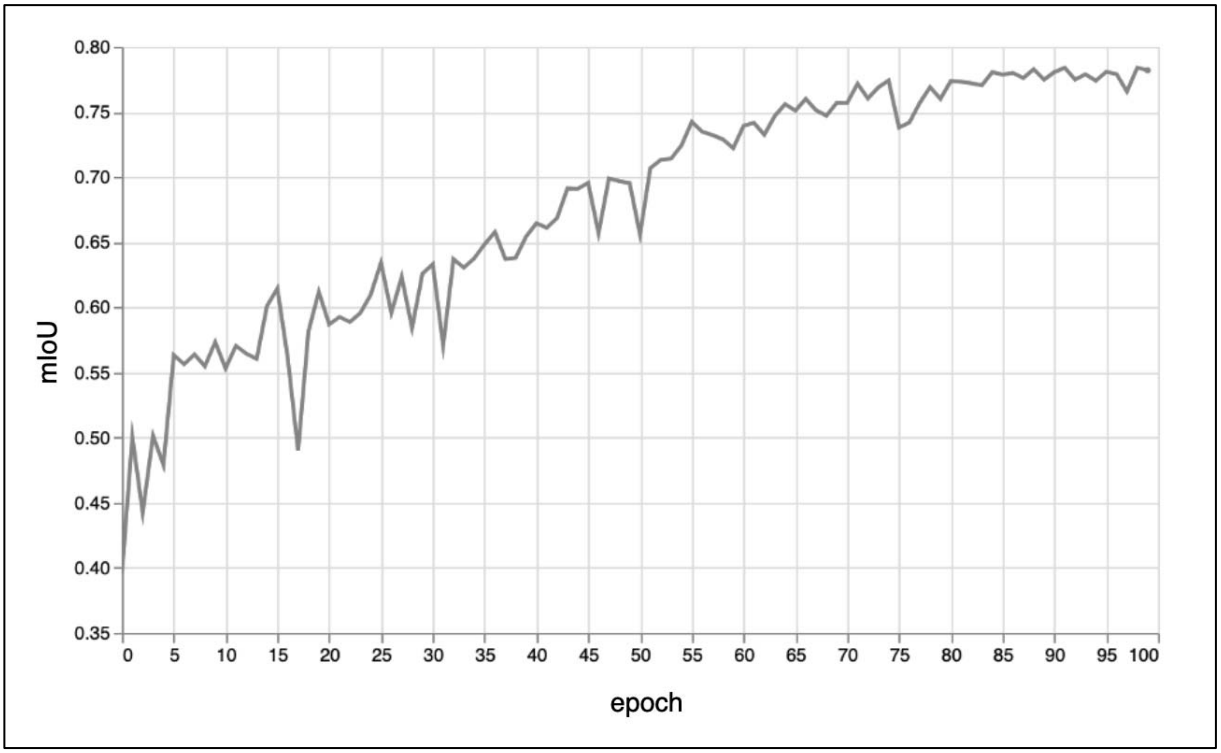

Рисунок 33 – Графики дистилляции нейронной сети

Выводы по четверной главе

В данной главе мы провели обучение нейронной сети для последующего редуцирования. Нейронная сеть обучалась на предварительно подготовленном наборе данных. Метрика mIoU нейронной сети на тестовой выборке составила 84,8%, метрика inference time составила 0,19 секунд, memory consumption 102 Мб. На рисунке 24 отображены графики метрики mIoU и ошибки loss на валидационной выборке в процессе обучения. На рисунке 1 приложения Б отображена топология архитектуры базовой нейронной сети.

Было проведено редуцирование нейронной сети. Сначала была получена более простая архитектура нейронной сети путем упрощения отдельных компонентов нейронной сети. Было проведено уменьшение энкодера до ResNet18. При уменьшении размера энкодера точность модели осталась на уровне исходной. При этом объем потребляемой памяти и время выполнения модели сократились примерно в три раза. Так же было проведено уменьшение других компонентов сети и в качестве лучшей модели будет взята

модель с 3мя входными каналами, и размером выходного слоя 256 и отключенными слоями batch-normalization в энкодере и декодере.

Затем было проведено квантование весов. Изменение типа данных в нейронной сети увеличило время обработки изображения до 2х секунд, а использование квантования после обучения снизило точность mIoU до 44%. Так же было проведено прореживание нейронной сети, однако при его проведении было обнаружено увеличение размера модели, а не ее уменьшение. Это связано с тем, что в файл модели проводится запись информации о нейронах, которые нужно инициализировать нулями, и уменьшение памяти менее заметно чем запись этой информации.

Была проведена факторизация трех слоев в нейронной сети. В следствии произошло уменьшение метрики mIoU и улучшение производительности. Для того, чтобы повысить точность нейронной сети была осуществлена дистилляция модели. Благодаря этому точность нейронной сети выросла для более простой модели.

В результате редуцирования была получена более простая модель, значение с метрики mIoU на тестовой выборке составила 84,3%, метрика inference time составила 0,03 секунд, memory consumption 4,5 Мб. На рисунке 2 приложения Б отображена топология архитектуры редуцированной нейронной сети.

ЗАКЛЮЧЕНИЕ

В рамках данной работы было проведено редуцирование нейронной сети для задачи сегментации. При этом были решены следующие задачи.

1. Проведен обзор архитектур.
2. Проведен обзор методов редуцирования нейронных сетей.
3. Проведен сбор и предобработка данных для обучения.
4. Обучена нейронная сеть и проведена оценка результатов ее работы.
5. Выполнено редуцирование нейронной сети.

В результате редуцирования была получена более простая модель, что привело к снижению потребления оперативной памяти. У исходной модели объем памяти составлял 102 Мб, в то время как у редуцированной модели этот показатель снизился до 9,8 Мб. Время обработки также уменьшилось: изначально оно составляло 0,19 секунды, а после редуцирования – 0,03 секунды. После проведения редуцирование значение метрики mIoU уменьшилось на 0,5%. В таблице 1 приложения В приведены все результаты проведенных экспериментов по редуцированию нейронной сети

В дальнейшем планируется разработка процесса редуцирования нейронных сетей, который будет перебирать различные комбинации методов оптимизации и осуществлять поиск наиболее оптимальной архитектуры по качеству и производительности.

ЛИТЕРАТУРА

1. ADE20K Dataset. [Электронный ресурс] URL: https://groups.csail.mit.edu/vision/datasets/ADE20K/index.html (дата обращения: 14.04.2024 г.).

2. Albumentations. [Электронный ресурс] URL: https://albumentations.ai/docs/api_reference/augmentations/transforms/ (дата обращения: 14.04.2024 г.).

3. Chen L., Papandreou G., Kokkinos I., Murphy K., Yuille A. Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs. // IEEE Transactions on Pattern Analysis and Machine Intelligence, 2018. – 834–848 pp.

4. Chen L., Zhu Y., Papandreou G., Schroff F. Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. // European Conference on Computer Vision, 2018. – 833–851 pp.

5. Cityscapes Dataset. [Электронный ресурс] URL: https://www.cityscapes-dataset.com (дата обращения: 14.04.2024 г.).

6. Cityscapes Test | PapersWithCode. [Электронный ресурс] URL: https://paperswithcode.com/sota/semantic-segmentation-on-cityscapes (дата обращения: 10.04.2024 г.).

7. Cityscapes Val | PapersWithCode. [Электронный ресурс] URL: https://paperswithcode.com/sota/semantic-segmentation-on-cityscapes-val (дата обращения: 10.04.2024 г.).

8. COCO Dataset. [Электронный ресурс] URL: https://cocodataset.org/#home (дата обращения: 14.04.2024 г.).

9. Depthwise-convolution | PapersWithCode. [Электронный ресурс] URL: https://paperswithcode.com/method/depthwise-convolution (дата обращения: 25.04.2024 г.).

10. DVC. [Электронный ресурс] URL: https://dvc.org (дата обращения: 14.04.2024 г.).

11. Fasad Dataset | Kaggle. [Электронный ресурс] URL: https://www.kaggle.com/datasets/balraj98/facades-dataset (дата обращения: 14.04.2024 г.).

12. Felzenszwalb P., Huttenlocher D. Efficient Graph-Based Image Segmentation. // International Journal of Computer, 2004. – 167–181 pp.

13. Flask. [Электронный ресурс] URL: https://flask.palletsprojects.com/en/2.1.x/ (дата обращения: 14.04.2024 г.).

14. Get started with GitLab CI/CD | GitLab. [Электронный ресурс] URL: https://docs.gitlab.com/ee/ci/ (дата обращения: 14.04.2024 г.).

15. Global unstructured | Pytorch. [Электронный ресурс] URL: https://pytorch.org/docs/stable/generated/torch.nn.utils.prune.global_unstructureu.html (дата обращения: 29.04.2024 г.).

16. Groupwise point convolution | PaperWithCode. [Электронный ресурс] URL: https://paperswithcode.com/method/groupwise-point-convolution (дата обращения: 29.04.2024 г.).

17. Gupta A., Jalote P. An Experimental Evaluation of the Effectiveness and Efficiency of the Test Driven Development. // IEEE Transactions on Software Engineering, 2017. – 285–294 pp.

18. He K., Gkioxari G., Dollar P., Girshick R. Mask R-CNN. // IEEE International Conference on Computer Vision, 2017. – 2980–2988 pp.

19. House Numbers Dataset. [Электронный ресурс] URL: http://ufldl.stanford.edu/housenumbers/ (дата обращения: 14.04.2024 г.).

20. Hsiao S., Tsai B. Efficient Computation of Depthwise Separable Convolution in MoblieNet Deep Neural Network Models. // IEEE International Conference on Consumer Electronics-Taiwan, 2021. – 1–2 pp.

21. Introduction to quantization on pytorch | Pytorch. [Электронный ресурс] URL: https://pytorch.org/blog/introduction-to-quantization-on-pytorch/ (дата обращения: 29.04.2024 г.).

22. Jacob B., Kligys S., Chen B., Zhu M. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. // IEEE Conference on Computer Vision and Pattern Recognition, 2018. – 2704–2713 pp.

23. Karimov A., Razumov A., Manbatchurina R. Comparison of UNet, ENet, and BoxENet for Segmentation of Mast Cells in Scans of Histological SlicesENet. // International Multi-Conference on Engineering, Computer and Information Sciences, 2020. – 544–547 pp.

24. Knowledge distillation tutorial | Pytorch. [Электронный ресурс] URL https://pytorch.org/tutorials/beginner/knowledge_distillation_tutorial.html (дата обращения: 29.04.2024 г.).

25. Kubernetes. [Электронный ресурс] URL: https://kubernetes.io (дата обращения: 14.04.2024 г.).

26. Long J., Shelhamer E., Darrell T. Fully Convolutional Networks for Semantic Segmentation. // IEEE Conference on Computer Vision and Pattern Recognition, 2015. – 3431–3440 pp.

27. Mobile phone buying guide | Samsung. [Электронный ресурс] URL: https://www.samsung.com/uk/mobile-phone-buying-guide/how-much-memory/ (дата обращения: 17.04.2024 г.).

28. Neogi D., Nataraj Das N., Deb S. FitNet: A deep neural network driven architecture for real time posture rectification. // International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies, 2021. – 354–359 pp.

29. Numpy. [Электронный ресурс] URL: https://numpy.org/doc/ (дата обращения: 14.04.2024 г.).

30. Pandas. [Электронный ресурс] URL: https://pandas.pydata.org/docs/ (дата обращения: 14.04.2024 г.).

31. Pascal Voc Dataset. [Электронный ресурс] URL: http://host.robots.ox.ac.uk/pascal/VOC/ (дата обращения: 14.04.2024 г.).

32. Pillow. [Электронный ресурс] URL: https://pillow.readthedocs.io/en/stable/ (дата обращения: 14.04.2024 г.).

33. Pointwise-convolution | PapersWithCode. [Электронный ресурс] URL: https://paperswithcode.com/method/pointwise-convolution (дата обращения: 25.04.2024 г.).

34. Products A10 GPU | Nvidia. [Электронный ресурс] URL: https://www.nvidia.com/de-de/data-center/products/a10-gpu/ (дата обращения: 14.04.2024 г.).

35. Pruning of neural networks | Habr. [Электронный ресурс] URL: https://habr.com/ru/articles/575520/ (дата обращения: 29.04.2024 г.).

36. Pytest. [Электронный ресурс] URL: https://pytest-docs-ru.readthedocs.io/ru/latest/ (дата обращения: 14.04.2024 г.).

37. Python-poetry. [Электронный ресурс] URL: https://python-poetry.org (дата обращения: 14.04.2024 г.).

38. Python. [Электронный ресурс] URL: https://www.python.org (дата обращения: 14.04.2024 г.).

39. Pytorch. [Электронный ресурс] URL: https://pytorch.org (дата обращения: 14.04.2024 г.).

40. Qu H., Han L., Hu Z., Sun X., Huang H., Bao S. Weakly Supervised Boundary Localization for Medical Image Segmentation. // Neural Networks, 2023. – 1–13 pp.

41. Quantization | Pytorch. [Электронный ресурс] URL: https://pytorch.org/docs/stable/quantization.html pytorch (дата обращения: 29.04.2024 г.).

42. Ren S., He K., Girshick R. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. // IEEE Transactions on Pattern Analysis and Machine Intelligence, 2017. – 1137–1149 pp.

43. RoboFlow. [Электронный ресурс] URL: https://roboflow.com/ (дата обращения: 10.04.2024 г.).

44. Ronneberger O., Fischer P., Brox T. U-Net: Convolutional Networks for Biomedical Image Segmentation. // Medical Image Computing and Computer-Assisted Intervention, 2015. – 234–241 pp.

45. Samudre P., Shwnde P.; Jaiswal V. Optimizing Performance of Convolutional Neural Network Using Computing Technique. // IEEE International Conference for Convergence in Technology, 2020. – 343–354 pp.

46. Sandler M., Howard A., Zhu M., Zhmoginov A. MobileNetV2: Inverted Residuals and Linear Bottlenecks. // Proceedings of the IEEE conference on computer vision and pattern recognition, 2018. – 4510–4520 pp.

47. Schmidhuber J. Deep learning in neural networks: An overview. // Neural Networks, 2015. – 85–117 pp.

48. Scikit-learn. [Электронный ресурс] URL: https://scikit-learn.org (дата обращения: 14.04.2024 г.).

49. Segmentation model pytorch | GitHub. [Электронный ресурс] URL: https://github.com/qubvel/segmentation_models.pytorch (дата обращения: 29.04.2024 г.).

50. Shah H., Vaswani A., Dash T., Hebbalaguppe R. Empirical Study of Data-Free Iterative Knowledge Distillation. // International Conference on Artificial Neural Networks, 2021. – 546–557 pp.

51. Tan K., Wang D. Compressing Deep Neural Networks for Efficient Speech Enhancement. // IEEE International Conference on Acoustics, Speech and Signal Processing, 2021. – 8358–8362 pp.

52. Tan M., Quoc V. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. // Journal of Computer and Communications, 2021. – 6105–6114 pp.

53. Tanaka .H, Kunin D. Pruning neural networks without any data by iteratively conserving synaptic flow. // Proceedings of the 34th International Conference on Neural Information, 2020. – 6377–6389 pp.

54. Tuan T., Khoa N., Quan T. ColorRL: Reinforced Coloring for End-to-End Instance. // IEEE Conference on Computer Vision and Pattern Recognition, 2021. – 16722–16731 pp.

55. Vivienne Sze V., Chen Y., Yang T. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. // Proceedings of the IEEE, 2017. – 2295–2329 pp.

56. Xiang H., Yan B. An edge detection algorithm based-on Sobel operator for images captured by binocular microscope. // International Conference on Electrical and Control Engineering, 2011. – 980–982 pp.

57. Yu C., Wang D., Zhou P. Model compression and acceleration for deep neural networks. // IEEE Signal Processing Magazine, 2018. – 126–136 pp.

58. Zhang X., Zhou X., Lin M., Sun J. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. // Proceedings of the IEEE conference on computer vision and pattern recognition, 2018. – 6848–6856 pp.

59. Zheng S., Lu J., Zhao H. Pyramid Scene Parsing Network. // Conference on Computer Vision and Pattern Recognition, 2021. – 6230–6239 pp.

60. Zheng S., Lu J., Zhao H. Rethinking Semantic Segmentation from a Sequence-to-Sequence Perspective with Transformers. // Conference on Computer Vision and Pattern Recognition, 2021. – 6877–6886 pp.

61. Zhou Z., Siddiquee R., Tajbakhsh N., Liang J. UNet++: A Nested U- Net Architecture for Medical Image Segmentation. // Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support, 2018. – 3–11 pp.

62. Новиков Ф.А. Учебно-методическое пособие по дисциплине «Анализ и проектирование на UML». // СПб: СПбГУ ИТМО, 2008. – 286 с.

ПРИЛОЖЕНИЯ

Приложение А. Настройка автоматизации экспериментов

Листинг 1 – Основные детали настройки файла gitlab-ci

```
build:
 stage: build
 image: docker:20.10.6
 tags:
   - nastya-runner-gpu
 rules:
   - changes:
     - Dockerfile
 script:
   - docker login -u $REGISTRY_USER -p $REGISTRY_PASSWORD <<host>>
   -   docker   build   -t   <<host>>/reduction-of-the-semantic-segmenter-
deps:latest -f Dockerfile .
   - docker push <<host>>/reduction-of-the-semantic-segmenter-deps:latest

test:
 stage: test
 image: <<host>>/reduction-of-the-semantic-segmenter-deps:latest
 tags:
   - nastya-runner-gpu
 script:
    - pytest

experimentation:
 stage: experimentation
 image: <<host>>/reduction-of-the-semantic-segmenter-deps:latest
 tags:
   - nastya-runner-gpu
 script:
    -   git   remote   rm   origin   &&   git   remote   add   origin
git@gitlab.com:nastasy822/reduction-of-the-semantic-segmenter.git
    - dvc config --global studio.token $DVC_STUDIO_TOKEN
    - dvc exp run --verbose

deploy:
  stage: deploy
  image: lwolf/helm-kubectl-docker:v1.21.1-v3.6.0
  dependencies:
    - build
  variables:
    KUBECONFIG: /etc/deploy/config
  environment:
    name: Development
  before_script:
    - echo ${KUBE_CONFIG} | base64 -d > ${KUBECONFIG}
  script:
    - helm version
    - RELEASE_NAME=segmantation-api
    - helm repo add bitnami https://charts.bitnami.com/bitnami
    - helm upgrade --install --namespace city-beauty --create-namespace --
kubeconfig ${KUBECONFIG}
      --values ./API/ci/values-custom.yaml
      --set "image.pullCredentials.registry=${FINAL_CACHE_TAG}"
      bitnami/nginx --version 4.4.7
  rules:
    - if: $CI_COMMIT_BRANCH == "main"
```

Листинг 2 – Dockerfile

```
FROM python:3.10.9-slim as base

ENV LANG C.UTF-8
ENV LC_ALL C.UTF-8

ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONFAULTHANDLER 1
ENV PYTHONUNBUFFERED 1

RUN python -m pip install --upgrade pip

ENV POETRY_VERSION=1.2.2
ENV POETRY_HOME='/opt/poetry'
ENV POETRY_VENV='/opt/poetry-venv'
ENV POETRY_CACHE_DIR='/opt/.cache'
ENV PYTHONPATH="${PYTHONPATH}:/app-workspace"

WORKDIR /app-workspace
COPY . /app-workspace

RUN python -m venv $POETRY_VENV \
    && $POETRY_VENV/bin/pip install -U pip setuptools \
    && $POETRY_VENV/bin/pip install poetry==${POETRY_VERSION}

ENV PATH="$POETRY_VENV/bin:$PATH"

RUN poetry config virtualenvs.create false
RUN poetry install --no-interaction --no-ansi
```

Листинг 3 – Конфигурационный файл DVC

```
data_paths:
  test: test.csv
  train: train.csv
  val: val.csv

base:
  mask_dir: dataset/label_images_semantic
  original_dir: dataset/original_images
  n_classes: 23
  batch_size: 3
  model_name: PSPNet.pt

dataset_preparer:
  test_size: 0.25

training:
  test_size: 0.25
  lr: 0.001
  epochs: 15
  weight_decay: 0.0001
```

Приложение Б. Топологии нейронных сетей

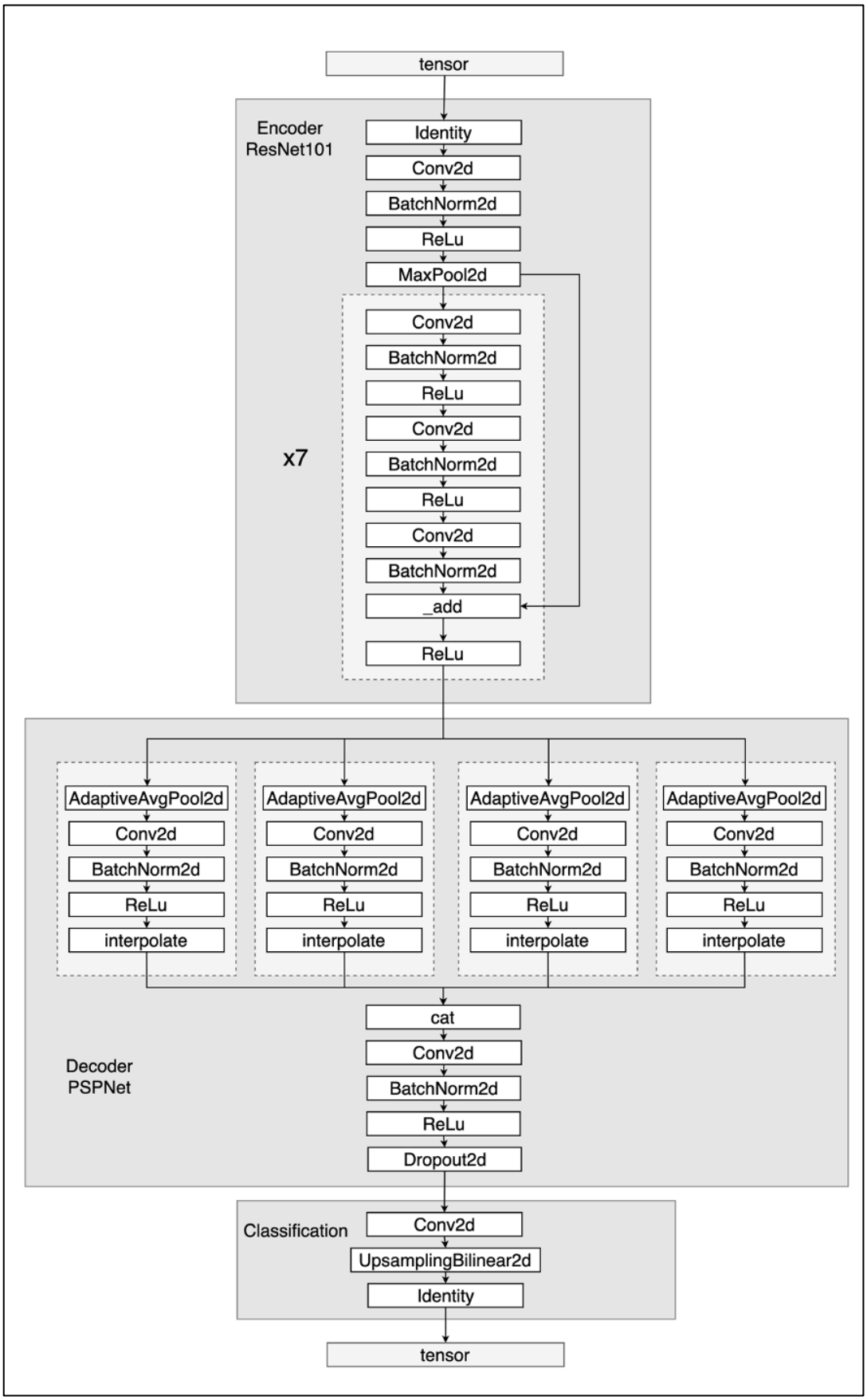

Рисунок 1 – Топология базовой архитектуры нейронной сети

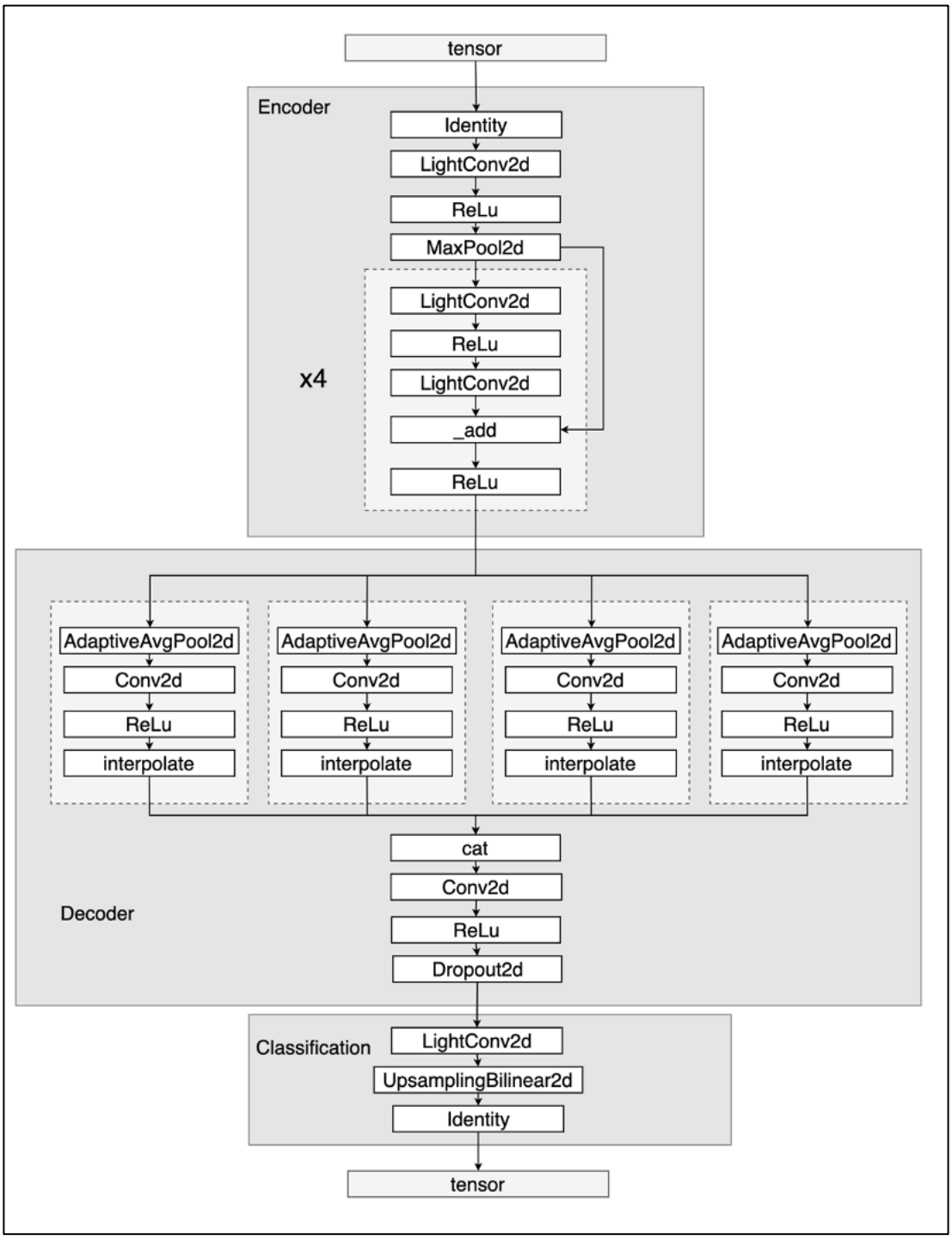

Рисунок 2 – Топология редуцированной архитектуры нейронной сети

Приложение В. Результаты проведенных экспериментов

Таблица 1 – Результаты проведенных экспериментов по редуцированию нейронной сети

Метод редуцирования	**mIoU (%)**	**inference time (сек)**	**memory consumption (Мб)**	**train time (сек)**
Использование энкодера ResNet101	84,8	0,19	102	105780
Использование энкодера ResNet50	86,1	0,17	102	25630
Использование энкодера ResNet34	85,7	0,07	43	3515
Использование энкодера ResNet18	85,2	0,06	35	3460
Использование энкодера ResNet18	85,2	0,06	35	3460
Снижение количества входных слоев до 1	77,8	0,05	35	2032
Удаление слоев batch-normalization из декодера	85,5	0,06	33	3207
Снижение размера выходного слоя до 256	85,3	0,04	33	3560
Снижение размера выходного слоя до 128	84,5	0,06	33	3582
Снижение глубины энкодера до 1	83,3	0,06	26	3012
Снижение глубины энкодера до 2	78,4	0,04	22	3367
Удаление слоев batch-normalization из энкодера	82,7	0,06	22	3697
Использование типа данных float16	–	3,56	17	–
Использование типа данных bfloat16	–	2,85	12	–
Квантование после обучения	44,8	0,04	11,65	3650
Факторизация сверточных слоев	73,7	0,03	4,5	3154
Дистилляция модели (15 эпох)	72,0	0,03	4,5	3246
Дистилляция модели (100 эпох)	84,3	0,03	4,5	22484

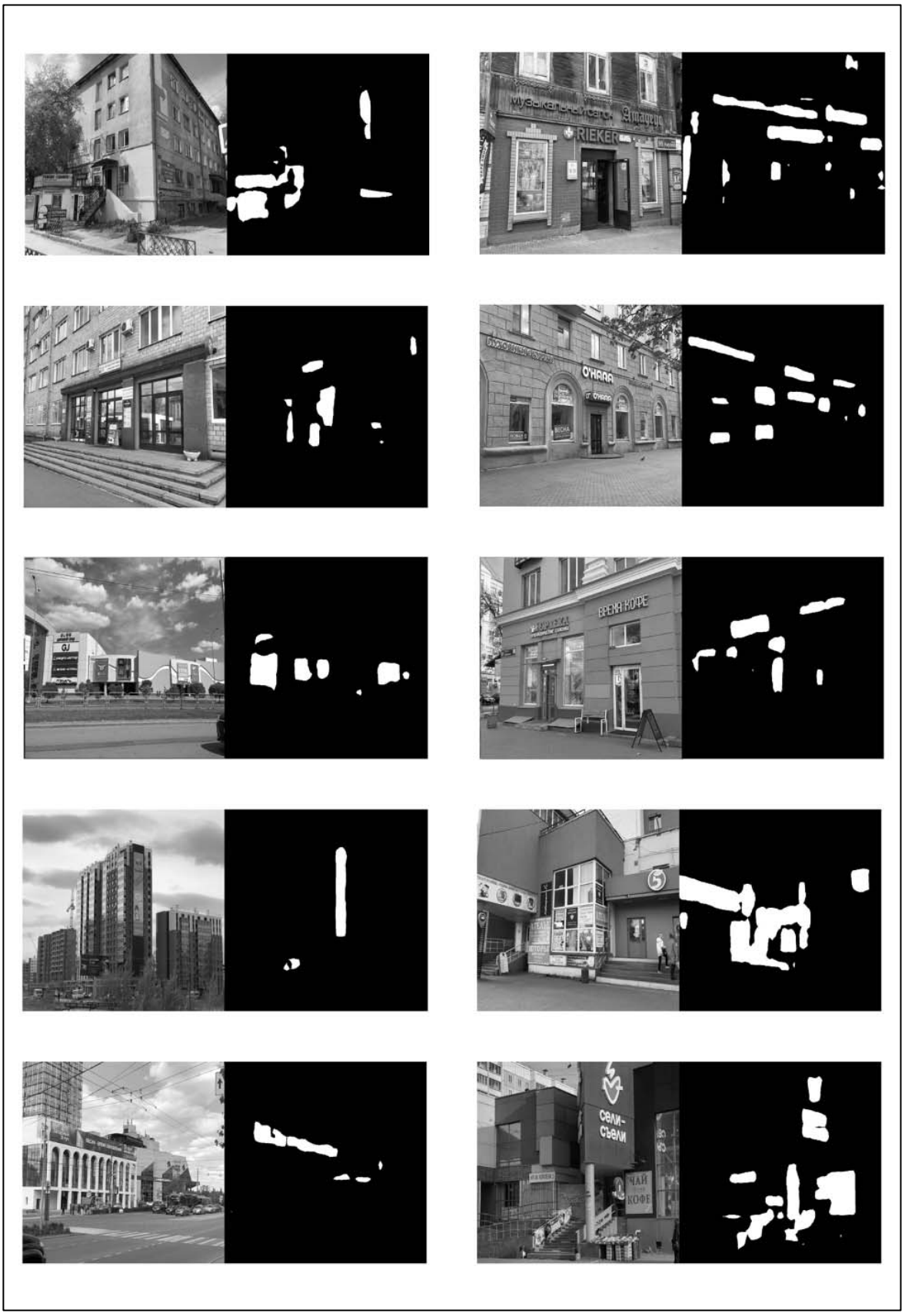

Рисунок 3 – Примеры сегментации рекламы
на изображениях фасадов зданий