

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего образования
«Пермский государственный национальный исследовательский университет»

Физико-математический институт

ОТЧЕТ

о прохождении технологической (проектно-технологической) практики

Направление/специальность: 01.04.02 Прикладная математика и информатика

Студент Группы ПМИ 1 курса

_____ **Ю.В. Санников**

Руководитель практики, директор
института, доктор физико-математических
наук, доцент:

_____ **М.А. Барулина**

Сроки прохождения практики с «01» июня 2025 г. по «24» июня 2025 г.

Пермь 2025 Год

АННОТАЦИЯ

Проектно-технологическая практическая работа 43 с., 29 рис., 3 табл., 15 источн., 1 прил.

Данная работа посвящена закреплению изученного материала и применению его на практике, для определения направления ИИ в котором я буду дальше развиваться.

Во введении представлено обоснование актуальности прохождения технологической практики и определения направления для выпускной работы на следующий год.

Первая глава содержит обзор изученных ранее в этом году дисциплин.

Во второй главе содержится описание применения полученных знаний на практике.

В третьей главе проводятся выводы по проделанной работе и выбор интересующего направления.

В заключении подведен итог проделанной работы и приведены перспективы развития.

Приложение А содержит ссылку на репозиторий с исходным кодом программы.

СОДЕРЖАНИЕ

ГЛОССАРИЙ	4
ВВЕДЕНИЕ	5
1. Анализ пройденных дисциплин	7
1.1. Препроцессинг изображений	7
1.2. Сегментирование с использованием нейронных сетей	9
1.3. Архитектуры U-Net и U-Net++.....	12
1.3.1. U-Net.....	13
1.3.2. U-Net++	16
1.4. Работа с YOLO	17
2. Применение знаний на практике	19
2.1. Сегментация изображений с использованием компьютерного зрения 20	
2.1.1. Функции для работы со свёрточными слоями	20
2.1.2. Реализация архитектуры U-Net	22
2.1.3. Реализация архитектуры U-Net++L3	23
2.2. Тестирование и выбор лучшей из сегментационных моделей	24
2.2.1. Результаты обучения	25
2.2.2. Результаты сегментации	30
2.2.3. Выбор лучшей модели	33
2.3. Использование YOLO для сегментации изображений	33
3. Результаты и выводы по проделанной работе.....	37
ЗАКЛЮЧЕНИЕ	38
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	39
Приложение А ссылка на репозиторий с кодом программы и результатами сегментации	42

ГЛОССАРИЙ

ИИ – искусственный интеллект.

Препроцессинг - это процесс предварительной обработки данных.

CNN – (Convolutional Neural Network) это тип алгоритма глубокого обучения, предназначенный для обработки визуальных данных.

Тензор - это представление данных, которое используется для решения задач сегментации изображений или временных рядов. Например, в задаче семантической сегментации изображений, элементы тензора отвечают о принадлежности каждого пикселя к определённому классу.

ВВЕДЕНИЕ

В рамках прохождения технологической практики важным аспектом является использование теоретических знаний с практическими навыками. Только изучая теорию, мы знакомимся с разными теориями и методами, но, чтобы научиться владеть полученными знаниями, нужно учиться применять их в конкретных ситуациях. Технологическая практика предоставляет уникальную возможность не только восполнить пробелы в знаниях, но и применить в реальных условиях. Именно в процессе решения поставленных реальных задач формируется глубокое понимание предмета.

Также, перед студентом стоит задача определить направление, в котором он собирается развиваться. Понимание направления и темы выпускной квалификационной работы является ключевым этапом его академического пути, так как это определяет фокус исследований и практической деятельности на заключительный год магистранта. Выбор темы должен отражать интересы студента, а также актуальные тренды и проблемы выбранной области. Четкое понимание темы позволит сосредоточиться на развитии необходимых навыков и компетенций, которые будут полезны как в процессе написания работы, так и в будущей профессиональной деятельности.

Объектом исследования являются дисциплины, связанные с искусственным интеллектом.

Предметом исследования методы для решения поставленных задач

Целью технологической практики является закрепление теоретических знаний, полученных при изучении дисциплин, развитие практических умений и навыков, формирование требуемых профессиональных компетенций, приобретение опыта в решении реальных профессиональных задач.

Для достижения поставленной цели необходимо было решить следующие задачи:

1. пройти по дисциплинам, изученным ранее в этом году и освежить материал,

2. изучить углубленно методики сегментации изображений способами компьютерного зрения,

3. разработать и обучить модели для сегментации изображений на основе искусственных нейронных сетей, существующих архитектур на примере U-Net и YOLO,

4. описать интересное направление ИИ, для написания выпускной квалификационной работы.

Реализация вышеперечисленных задач будет способствовать достижению цели технологической практики.

Инструменты, использованные при разработке программы: Python, Tensorflow, CUDA, Google Colab.

1. Анализ пройденных дисциплин

Пройденными дисциплинами, связанными с искусственным интеллектом, являются:

- интеллектуальный анализ данных методами машинного обучения,
- системы искусственного интеллекта,
- компьютерное зрение.

Самый актуальный предметом из перечисленных, компьютерное зрение, будет рассмотрен в рамках первого пункта технологической практики, так как он содержит в себе знания, полученные в рамках остальных дисциплин.

В рамках дисциплины «Компьютерное зрение», происходило знакомство с методами работы с изображением, архитектурой сверточных сетей, способами сегментации изображения, методами работы с объектами на изображении, работа с видео для обнаружения поз (использование технологий Pose Detection).

Далее будут рассмотрены:

- препроцессинг изображений,
- сегментирование с использованием нейронных сетей,
- архитектуры Unet и Unet++,
- работа с YOLO,
- технологии обнаружения поз на видео.

1.1. Препроцессинг изображений

В рамках дисциплины «Компьютерное зрение», происходило знакомство с методами работы с изображением. В python используется OpenCV - открытая библиотека для работы с алгоритмами компьютерного зрения, использующая оптимизированные библиотеки на C++ и предоставляет API, с использованием массивов numpy [1]. Для чтения, записи и фильтров в среду разработки импортируется библиотека cv2. Фильтр для обработки изображений представляет собой метод, при котором каждый пиксель

изображения последовательно обрабатывается с помощью оператора, представленного в виде квадратной матрицы. Результатом работы этого метода является оценка значимости каждого пикселя изображения.

Рассмотрим некоторые функции для работы с изображением, использовавшиеся во время практики:

- `cv2.imread (path, cv2.IMREAD_GRAYSCALE)` чтение картинки, по указанному пути, в данной работе изображения сразу считывались как черно-белые с помощью `cv2.IMREAD_GRAYSCALE`;
- `cv2.resize (img, x, y)` – изменяет масштаб картинки, принимает на вход изображение и аргументы ширины с высотой;
- `cv2.GaussianBlur (img, (13,13), 0)` – размывает изображение, вторым аргументом подается квадратная матрица (ядро свертки), где пиксель в центре матрицы устанавливается как взвешенное среднее значение у соседних пикселей, чем больше матрица, тем больше размытие, третьим аргументом устанавливается стандартное отклонение ядра Гаусса;
- `cv2.threshold(blur, 100, 255, cv2.THRESH_BINARY)` - позволяет применять порог к каждому пикселю изображения. Она принимает четыре параметра: исходное изображение, значение порога, максимальное значение и тип thresholding, в работе используется порог `cv2.THRESH_BINARY`, пиксели выше порога получают максимальное значение (255 – белый), пиксели ниже порога получают значение 0 (черный), в рамках текущей работы, блюр изображения с помощью `GaussianBlur`, в связке с применением фиксированного порога используются, чтобы убрать лишние шумы с предсказанной маски;
- `cv2.addWeighted(mask, alpha, original_image, 1-alpha, 0, original_image)` – используется для наложения на предсказанное изображение исходного с некоторой прозрачностью, используется в работе модели, для предсказания маски у единственного изображения;
- `cv2.cvtColor(img, cv2.COLOR_BGR2RGB)` - преобразует изображение из одного цветового пространства в другое, так-как в OpenCV

формат цвета по умолчанию имеет формат цвета BGR, для корректной работы рекомендуется преобразовать его в RGB с помощью `cv2.COLOR_BGR2RGB`;

- `cv2.imshow(img, name)` - создаёт окно с заданным именем и отрисовывает в нём изображение;

- `cv2.imwrite (save_image_path, img)` – сохраняет изображение по указанному пути.

1.2. Сегментирование с использованием нейронных сетей

С развитием нейросетей за последнее десятилетие, методы анализа изображений переживают ренессанс. Нейросетевые подходы позволили повысить уровень точности и эффективности в задачах компьютерного зрения. Новые методы превосходят традиционные в сложных задачах сегментации изображения и видео. Задача сегментации объекта на картинке, представляет собой классификацию по пикселям, определение принадлежности пикселя объекту или фону.

Современные методы сегментации изображений обычно адаптированы с архитектур свёрточных нейронных сетей (CNN – Convolution Neuro Network) или трансформеров [2]. Существует множество архитектур, разработанных для решения данной задачи. К наиболее распространенным архитектурам относятся:

- Сети U-Net, которые будут подробно рассмотрены в следующем пункте;

- DeepLab (2015) [3] архитектуры, используют ASPP (Atrous Spatial Pyramid Pooling) свёрточные сети, улавливающие контекст на разных масштабах, последние версии DeepLabV2 и DeepLabv3+ (2021) позволяют более точно восстанавливать пространственную информацию, выводя ее в число ведущих отраслей в этой сфере. Общая схема метода DeepLabV3 и DeepLabv3+ представлена на рисунке 1.

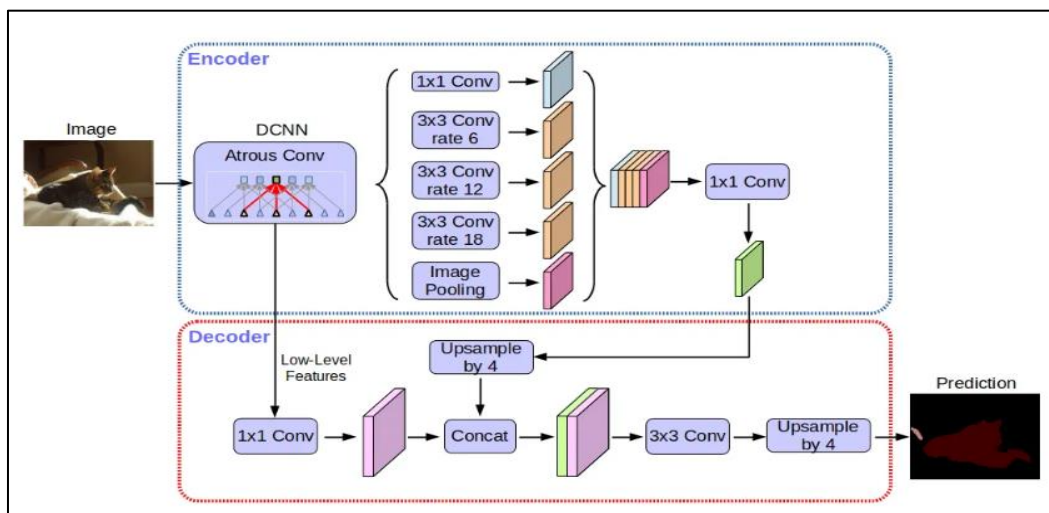


Рисунок 1. Общая схема метода DeepLabV3 и DeepLabv3+.

– Mask R-CNN (2017) [4], представляет собой улучшенную версию R-CNN (Region-based Convolutional Neural Network) [5]. Главное новшество Mask R-CNN заключается в внедрении механизма сегментации на уровне масок, который позволяет точно определить границы объектов, что делает возможным не только выделение классов объектов, но и их формы. Общая схема метода Mask R-CNN представлена на рисунке 2.

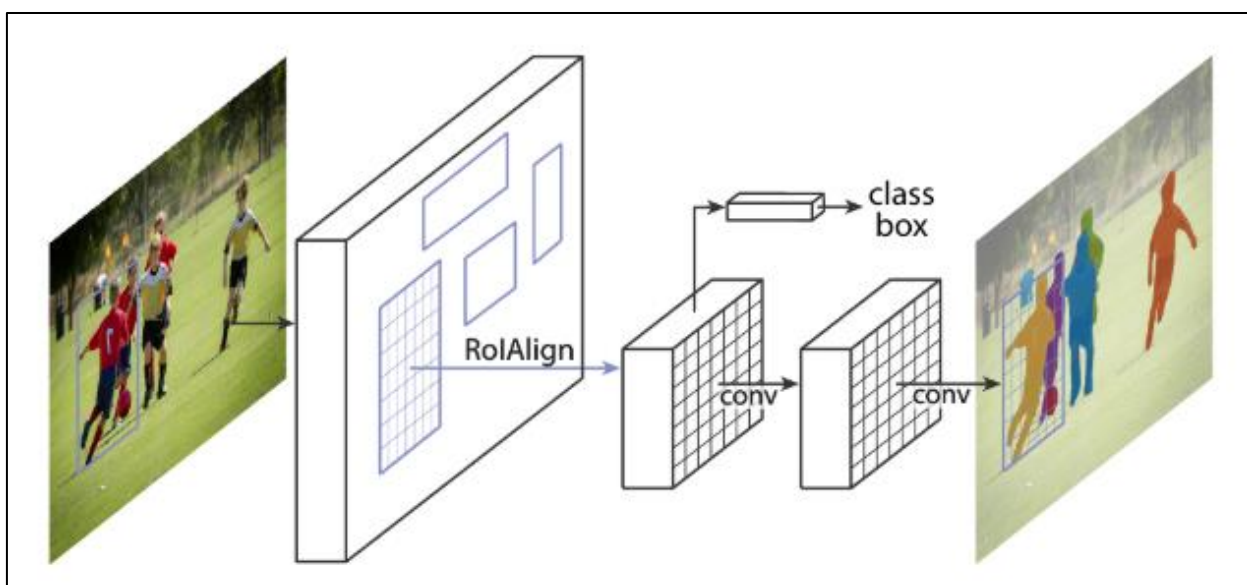


Рисунок 2. Общая схема метода Mask R-CNN.

– SegNet (2015) [6]. Была разработана для того, чтобы эффективно решать задачи сегментации с относительно низкими вычислительными затратами, особенно для приложений, требующих высоких разрешений изображений. Не смотря на свою эффективность и оптимизированность, главным недостатком такой модели является ее точность. Схема архитектуры SegNet представлена на рисунке 3.

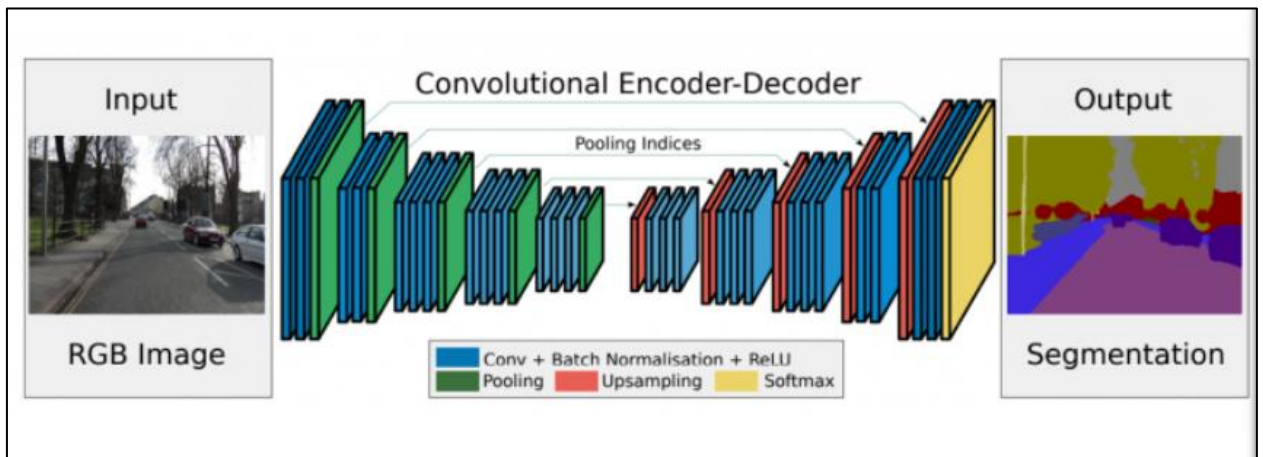


Рисунок 3. Общая схема архитектуры SegNet.

– Transformer-базированные архитектуры ViT (VisionTransformer) (2021) [7] представляют изображения в виде последовательных патчей 16x16 пикселей, которые равномерно преобразуются в векторные представления и подаются на вход трансформеру. В отличие от CNN, ViT не использует свертки, что позволяет захватывать более обширные контексты. К недостаткам метода можно отнести необходимость большого количества данных для обучения. Схема архитектуры Visual Transformer проиллюстрирована на рисунке 4.

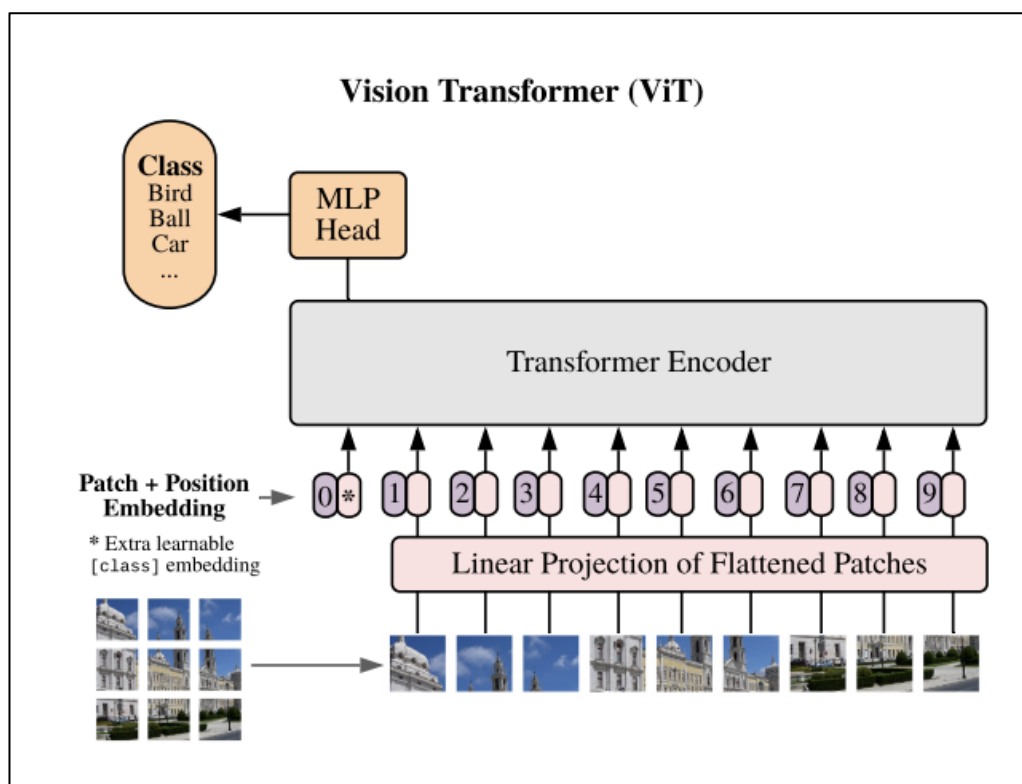


Рисунок 4. *Схема архитектуры Visual Transformer.*

Основными недостатками описанных выше методов, являются чрезмерные вычислительные затраты из-за сложных архитектур.

1.3. Архитектуры U-Net и U-Net++

U-Net является одной из стандартных архитектур CNN для задач сегментации изображений, использующаяся, для сегментации области изображения по классам. Для U-Net характерно получение хороших результатов в различных реальных задачах, при использовании небольшого количества данных. Данный метод был выбран для углубленного изучения, в связи с простотой построения архитектуры и ее понимания.

Будут рассмотрены:

- U-Net,
- U-Net++.

1.3.1. U-Net

Архитектура U-Net, изначально разработанная Олафом Роннебергером и др. в 2015 году [8], использовалась для медицинских целей в сегментации биомедицинских изображений. Свое название она получила, из-за U-образной формы схемы модели, представленной на рисунке 5.

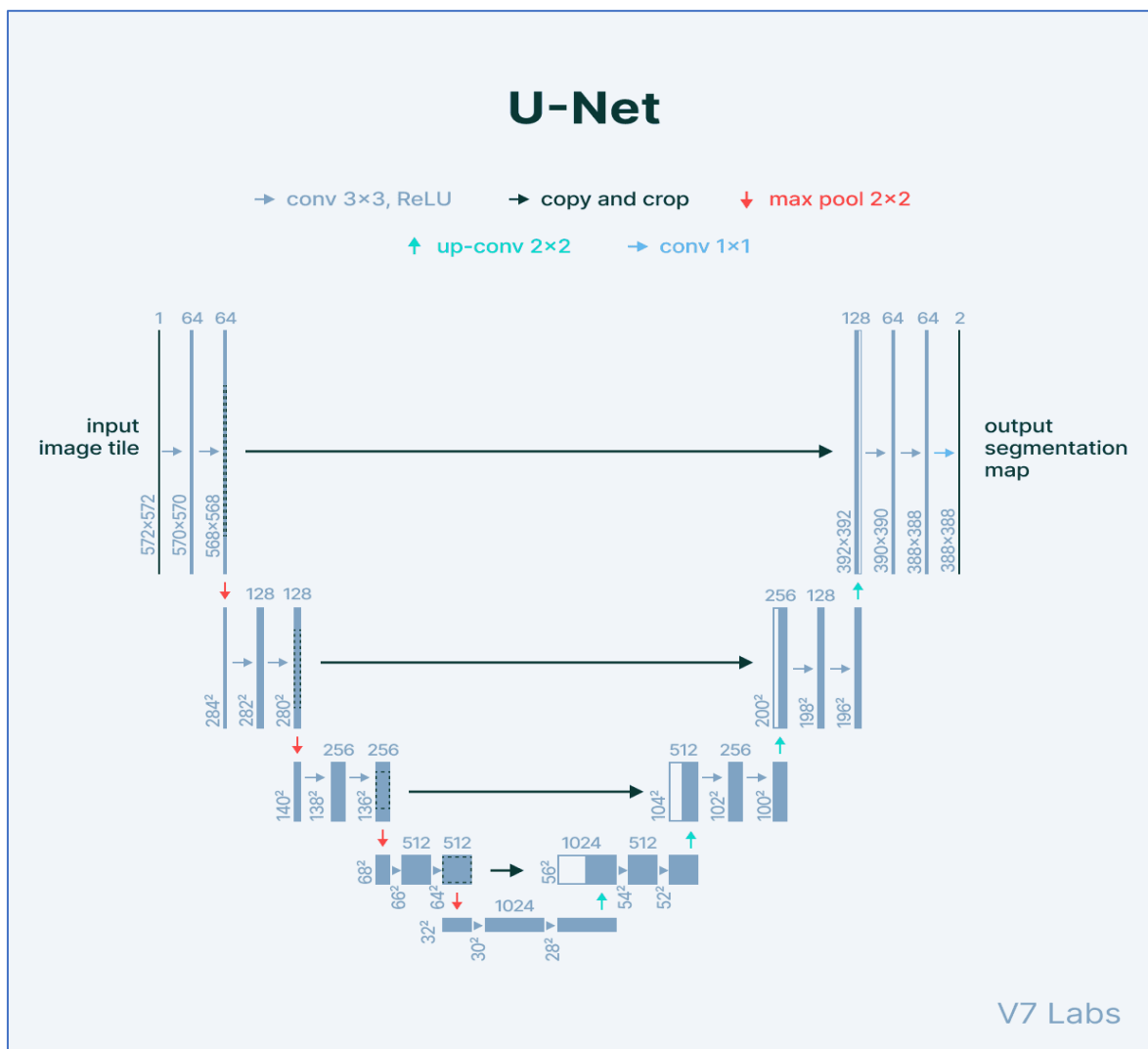


Рисунок 5. Схема архитектуры U-Net.

На рисунке 5, в левой части схемы располагается свёрточная сеть – Encoder, в правой стороне Decoder – расширяющийся путь [9]. Ключевой идеей схемы, заключается в том, что каждый промежуточный результат encoder-а конкатенируется с промежуточным результатом правой части архитектуры. Такое решение (skip-connection) было представлено для решения проблемы потери информации во время операции max-pooling.

Encoder работает следующим образом, если на вход подается изображение $572 \times 572 \times 1$ и с помощью операции свертки (convolution 3×3) с функцией активации ReLU, и нулевым отступом, для предотвращения потери информации на границах. Слой, создаваемый сверткой, позволяет изучать более локальные признаки, например, обнаружение глаза на изображении. Пример свертки 3×3 представлен на рисунке 6.

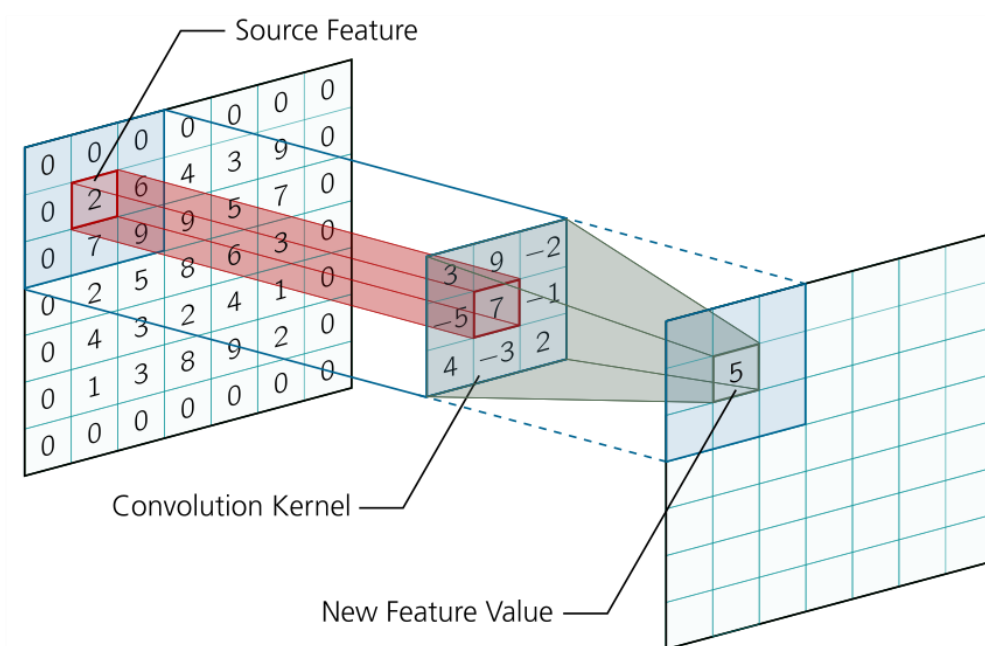


Рисунок 6. Пример свертки 3×3 .

После данных манипуляций получаются тензоры $572 \times 572 \times 64$. Следующим шагом применяется maxPooling 2×2 , уменьшающий размер тензора по ширине и высоте вдвое. Помимо того, что данный шаг помогает уменьшить количество параметров модели, тем самым предотвратив переобучение, он также повышает надежность модели, предотвращая неточности при небольших изменениях во входных данных. Суть метода maxPooling 2×2 представлена на рисунке 7.

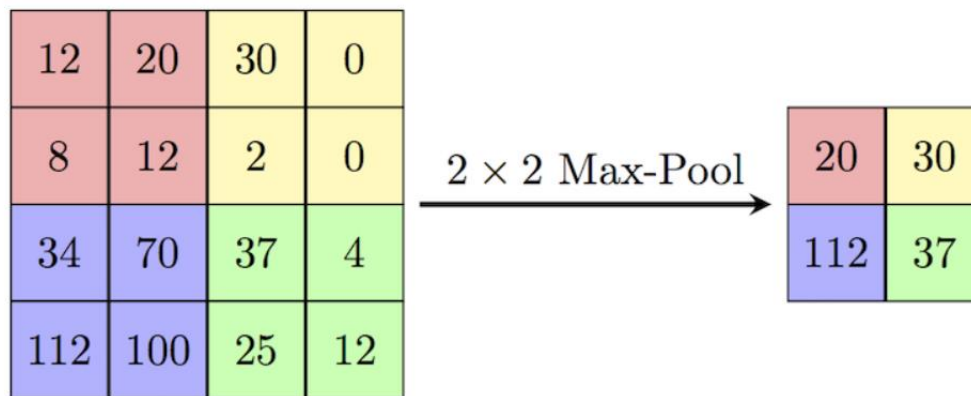


Рисунок 7. Пример *maxPooling 2x2*.

Эти два шага повторяются до момента, когда тензор не станет размером $28 \times 28 \times 1024$.

Основной целью декодера, является реконструкция карты сегментации высокого разрешения из изученных признаков, тем самым восстановив пространственную детализацию. Так как при *max-pooling* терялась важная информация, при выполнении восстановления изображения с помощью *up-sampling* результат получался не точным, из-за потери контуров объектов, поэтому для приведения тензора к целевому размеру используется *up-convolution 2x2* или *conv2dTranspose 2x2*, после чего берется тензор 64×64 с предыдущего этапа и конкатенируется с полученным результатом *up-conv*, чтобы размеры тензоров совпадали, от предыдущий тензор обрезают до 56×56 . Данная операция позволяет восстановить признаки, утраченные во время сжатия до тензора 28×28 . На рисунке 8 представлена часть схемы с переходом от свёрточной части к восстанавливающей.

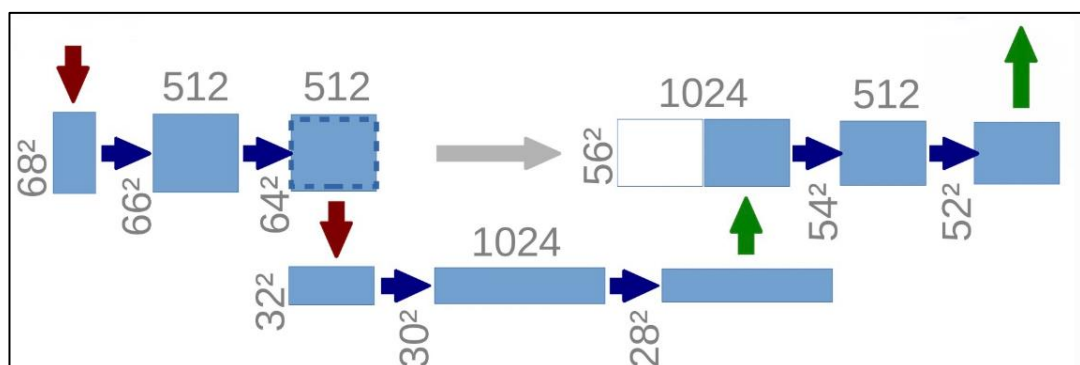


Рисунок 8. Переход от *encoder* части к *decoder*.

1.3.2. U-Net++

U-Net++, представленная в 2018 году является модификацией существующей модели U-Net [10]. Для улучшения показателей традиционной U-Net было предложено использование плотных пропускных соединений (skip-connections). На рисунке 9 представлена схема U-Net++.

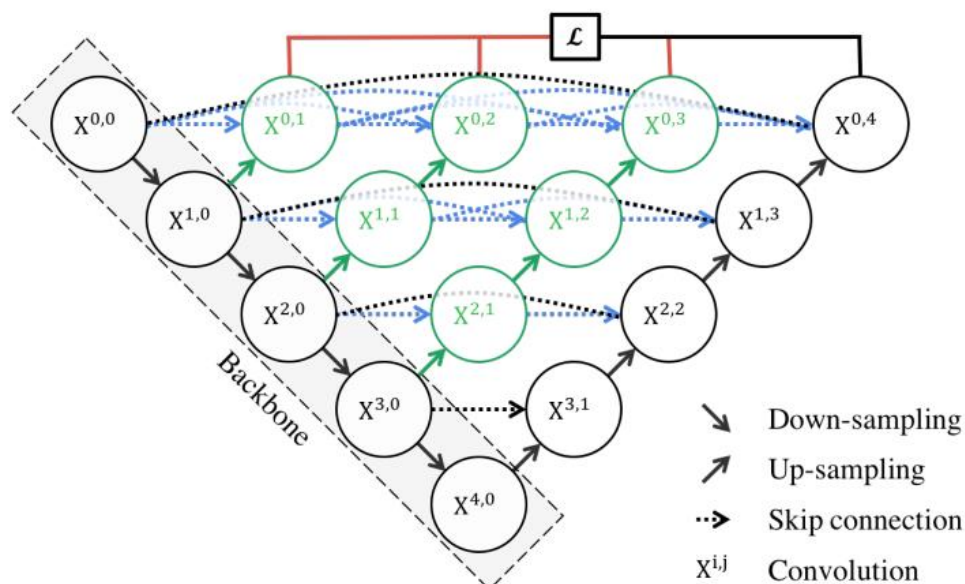


Рисунок 9. Архитектура U-Net++.

Обновленный дизайн архитектуры предлагает проводить up-convolution с предыдущими параметрами на каждом шаге свертки, полученный skip-connection передается далее, пока не упирается в декодер. Каждый такой мост строится за счет соединения (concatenate) предыдущих узлов up-sampling с свертыванием на пропускном соединении (рисунок 10). Такой подход позволяет более точно сократить семантический разрыв при восстановлении изображения.

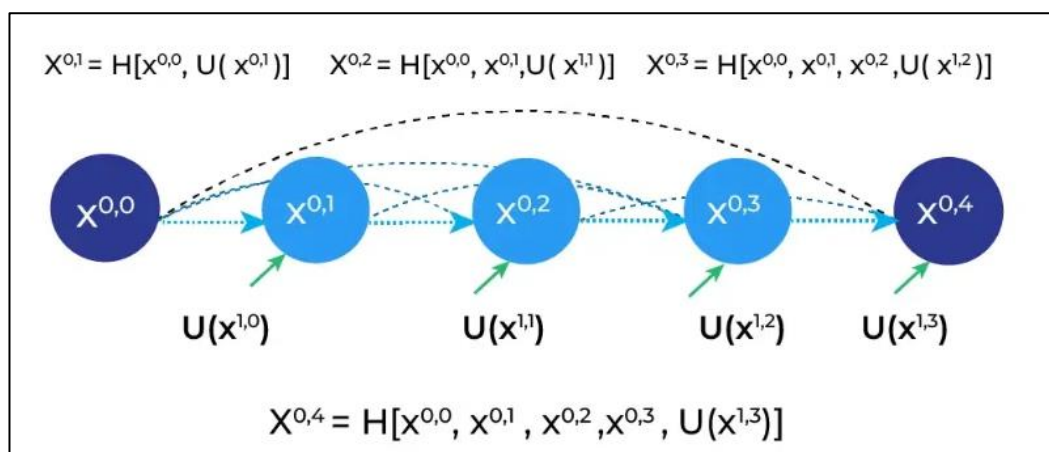


Рисунок 10. Оптимизация skip-connection с использованием комбинирования узлов.

Количество шагов вниз и вверх в этих моделях обозначаются буквой L . Экспериментальным путём было доказано, что архитектура U-Net++ L3 при только 3-х слоях обучается на 32,2% быстрее чем U-Net++L4, при минимальном отклонении в итоговом результате [11]. Схема U-Net++L3 представлена на рисунке 11.

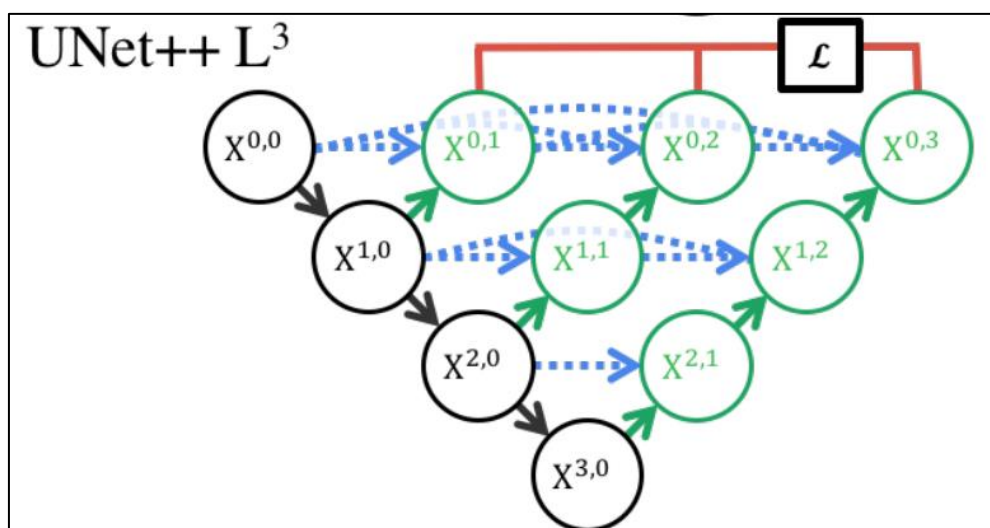


Рисунок 11. Архитектура U-Net++L3.

1.4. Работа с YOLO

YOLO (You Only Look Once) - универсальный фреймворк искусственного интеллекта, который поддерживает множество задач

компьютерного зрения [12]. С помощью фреймворка можно выполнять обнаружение, сегментацию, классификацию и оценку позы.

Первая версия YOLO (YOLOv1) была представлена в 2015 году командой исследователей во главе с Джозефом Редмоном [13]. В отличие от традиционных методов, YOLO рассматривает обнаружение объектов как единую задачу регрессии, что позволило ускорить обработку и сделать модель подходящей для работы в режиме реального времени.

Преимущества YOLO:

- Высокая точность: в сегментации используется модификация архитектуры U-Net, что позволяет достигать точных результатов.
- Скорость: YOLO оптимизирован для работы в реальном времени, обеспечивая быструю обработку изображений с высоким разрешением.
- Широкий спектр применения: он отлично подходит для медицинской визуализации, автономного вождения и других задач, где требуется детальная сегментация изображений.

Сайт Ultralytics предлагает заранее обученные модели Yolo, можно выбрать модели в зависимости от скорости и качества обработки изображения. Выбор модели определяется задачей, если нужна обработка в реальном времени, берется модель с самой быстрой обработкой, иначе берется модель с медленной обработкой изображения, но с высоким качеством результата.

Модель YOLO можно дообучать с использованием сервиса Roboflow, который предназначен для аннотирования обучающих наборов данных в задаче обнаружения объектов. Этот инструмент позволяет расширять уже имеющиеся датасеты за счет добавления новых данных. Публичные датасеты с Roboflow легко импортируются в среду разработки.

2. Применение знаний на практике

В качестве среды разработки использовался Google Colab - облачный сервис от Google, который позволяет писать и выполнять Python-код в браузере без установки дополнительных программ [14]. В Colab есть возможность подключения к удаленной среде выполнения с графическим аппаратным ускорением T4, что позволяет использовать библиотеку Tensorflow для обучения нейросетевых моделей. Характеристики графического процессора представлены на рисунке 12.

NVIDIA-SMI 550.54.15			Driver Version: 550.54.15			CUDA Version: 12.4		
GPU	Name		Persistence-M	Bus-Id	Disp.A	Volatile	Uncorr.	ECC
Fan	Temp	Perf	Pwr:Usage/Cap		Memory-Usage	GPU-Util	Compute M.	
							MIG	M.
0	Tesla T4		Off	00000000:00:04.0	Off			0
N/A	46C	P8	9W / 70W	0MiB / 15360MiB		0%	Default	N/A

Processes:								
GPU	GI	CI	PID	Type	Process name		GPU Memory	
	ID	ID					Usage	
No running processes found								

Рисунок 12. Характеристики графического процессора T4.

В качестве файлового менеджера был выбран Google-disk, так-как Google Colab не сохраняет переменные и файлы текущего сеанса после выхода. Функции для работы с обучением модели, были импортированы из библиотеки tensorflow.keras. К таковым относятся ModelCheckpoint, ReduceLROnPlateau, CSVLogger и т.д. из tensorflow.keras.callbacks.

- сегментация изображений с использованием компьютерного зрения,
- использование YOLO для сегментации изображений,

2.1. Сегментация изображений с использованием компьютерного зрения

Во время обучения, для сегментации изображений применялись U-Net архитектуры, в рамках текущей практики была реализована архитектура U-Net++L3. В качестве формулировки задачи, использовалась формулировка задачи с НИР, а именно выявление лабораторного животного (мыши) в сложной экспериментальной среде, то есть необходимо разработать программу, для определения каждого пикселя на принадлежность объекту животного на фото, выделение животного одним цветом, фона другим.

2.1.1. Функции для работы со свёрточными слоями

Для свертки модели U-Net++ conv_block_plus_plus. В функции дважды добавляются три слоя - свертка с ядром 3x3 и одинаковыми отступами по краям картинки, батч-нормализация (слой, нормализующий входные данные сети, для оптимизации обучения модели и борьбы с переобучением за счет стандартизации входов каждого из слоев) и функция активации, в случае с U-Net это relu. На рисунке 13 представлена реализация.

```
def conv_block_plus_plus(self, inputs, num_filters):
    x = tf.keras.Sequential([
        # Convolutional Layer
        Conv2D(num_filters, 3, padding='same'),
        # Batch Normalization Layer
        BatchNormalization(),
        # Activation Layer
        Activation('relu'),
        # Convolutional Layer
        Conv2D(num_filters, 3, padding='same'),
        # Batch Normalization Layer
        BatchNormalization(),
        # Activation Layer
        Activation('relu')
    ])(inputs)
    return x
```

Рисунок 13. Функции для свертки в архитектуре моделей U-Net++.

Описанные функции используются в блоках encoder и в decoder. Шаг сжатия изображения описан в функции encoder_block. Для U-Net он состоит

из конволюции предыдущего слоя с помощью `self.conv_block` и `MaxPool` с ядром `2x2`. Реализация для U-Net++ отличается тем, что `MaxPool` производится с использованием `strides(2,2)`, шаг смещения равен 2 по x и y вместо 1. На рисунке 14 описана функция `encoder_block`.

```
def encoder_block(self, inputs, num_filters,
isPlusPlus = False):
    if isPlusPlus:
        x = self.conv_block_plus_plus(inputs,
num_filters)
        p = MaxPool2D((2, 2), strides=(2, 2))(x)
        return x, p
    else:
        x = self.conv_block(inputs, num_filters)
        p = MaxPool2D((2, 2))(x)
        return x, p
```

Рисунок 14. Функция `encoder_block`.

Decoder, восстанавливающий пространство изображения, увеличивая его разрешение, описан в функции `decoder_block`. Для U-Net он состоит из транспонирования предыдущего слоя с шагом `(2,2)` и одинаковыми отступами, далее происходит конкатенирование с промежуточным результатом правой части, для восстановления потерянных при свертке данных, в конце шага с помощью `self.conv_block` происходит свертка. Реализация для U-Net++ отличается тем, что вместо транспонирования используется `upSampling` с ядром `(2,2)` для увеличения размера изображения. На рисунке 15 описана функция `decoder_block`.

```

def decoder_block(self, inputs, skip_features,
num_filters, isPlusPlus = False):
    if isPlusPlus:
        x = UpSampling2D((2, 2))(inputs)
        toConcat = skip_features + [x]
        x = concatenate(toConcat)
        x = self.conv_block_plus_plus(x, num_filters)
    else:
        x = Conv2DTranspose(num_filters, (2, 2),
strides=2, padding='same')(inputs)
        x = concatenate([x, skip_features])
        x = self.conv_block(x, num_filters)
    return x

```

Рисунок 15. Функция *decoder_block*.

2.1.2. Реализация архитектуры U-Net

Согласно архитектуре, обычный U-Net состоит из 4 *encoder_block* и 4 *decoder_block*. В конце вызывается Conv2D с ядром (1,1) одинаковыми отступами и сигмоидальной функцией активации. На рисунке 16 представлена реализация U-Net архитектуры.

```

def build_unet(self, input_shape):
    tf.keras.backend.clear_session()
    inputs = Input(input_shape)

    """ Encoder """
    s1, p1 = self.encoder_block(inputs, 64)
    s2, p2 = self.encoder_block(p1, 128)
    s3, p3 = self.encoder_block(p2, 256)
    s4, p4 = self.encoder_block(p3, 512)
    b1 = self.conv_block(p4, 1024)

    """ Decoder """
    d1 = self.decoder_block(b1, s4, 512)
    d2 = self.decoder_block(d1, s3, 256)
    d3 = self.decoder_block(d2, s2, 128)
    d4 = self.decoder_block(d3, s1, 64)

    """ Output """
    outputs = Conv2D(1, (1, 1), padding="same",
activation="sigmoid")(d4)

    return Model(inputs, outputs, name=self.unet)

```

Рисунок 16. Реализация U-Net архитектуры.

2.1.3. Реализация архитектуры U-Net++L3

Для реализации архитектуры U-Net++L3 использовались 3 encoder_block и 3 decoder_block. В отличие от обычной U-Net для получения более точных промежуточных результатов добавились переменные s1_0, s2_0 и s1_1. В эти переменные записывались результаты decoder_block со следующего шага и для конкатенации в декодере передавался массив из

полученных ранее на шаге промежуточных результатов. На рисунке 17 представлена реализация U-Net++L3 архитектуры.

```
def build_unet_plus_plus_modified(self, input_shape):
    tf.keras.backend.clear_session()

    inputs = Input(input_shape)
    inputs = Lambda(lambda x: x/255)(inputs)
    """ Encoder """
    s1, p1 = self.encoder_block(inputs, 64, True)
    s2, p2 = self.encoder_block(p1, 128, True)
    s3, p3 = self.encoder_block(p2, 256, True)
    b1 = self.conv_block_plus_plus(p3, 512)

    """ Inbetween """
    s1_0 = self.decoder_block(s2, [s1], 64, True)
    s2_0 = self.decoder_block(s3, [s2], 128, True)

    s1_1 = self.decoder_block(s2_0, [s1, s1_0], 64,
True)

    """ Decoder """
    d1 = self.decoder_block(b1, [s3], 256, True)
    d2 = self.decoder_block(d1, [s2, s2_0], 128,
True)
    d3 = self.decoder_block(d2, [s1, s1_0, s1_1],
64, True)
    """ Output """
    outputs = Conv2D(1, (1, 1), padding="same",
kernel_initializer = 'he_normal',
activation="sigmoid")(d3)
    return Model(inputs, outputs,
name=self.unetPlusPlusModified)
```

Рисунок 17. Реализация U-Net++L3 архитектуры.

2.2. Тестирование и выбор лучшей из сегментационных моделей

Будут рассмотрены:

- результаты обучения,
- результаты сегментации,

– выбор лучшей модели.

В качестве результатов обучения будут представлены результаты метрик и их сравнение. По результатам сегментации будет выбрана лучшая модель для сегментации видео. Будет приведен пример сегментации на изображении, представленном на рисунке 18.



Рисунок 18. *Изображение для сегментации.*

2.2.1. Результаты обучения

Результаты метрик, собранных во время обучения модели U-Net представлены в таблице 1.

Таблица 1. Результаты метрик тестовой и валидационной выборок после обучения на U-Net.

epoch	learning_rate	loss	mean_io_u	precision	recall	val_loss	val_mean_io_u	val_precision	val_recall
0	1.00E-04	-161.756	0.50122	0.15727	0.93592	2.989	0.4999	0.3021	0.2768
1	1.00E-04	-215.924	0.50118	0.41899	0.97399	-134.376	0.5027	0.7004	0.8638
2	1.00E-04	-239.651	0.50156	0.53821	0.98306	-201.896	0.5033	0.7711	0.9486
3	1.00E-04	-260.273	0.50159	0.60149	0.98568	-264.845	0.5026	0.6837	0.9727
4	1.00E-04	-281.227	0.50184	0.64916	0.98823	-280.885	0.5032	0.7618	0.9749
5	1.00E-04	-302.147	0.50196	0.67874	0.99040	-308.033	0.5034	0.7724	0.9719
6	1.00E-04	-323.050	0.50199	0.69510	0.99166	-325.307	0.5028	0.7778	0.9733
7	1.00E-04	-344.389	0.50205	0.70646	0.99310	-373.524	0.5024	0.7467	0.9769
8	1.00E-04	-366.196	0.50210	0.72051	0.99389	-417.529	0.5008	0.6795	0.9883
9	1.00E-04	-388.613	0.50222	0.73570	0.99478	-421.650	0.5002	0.6556	0.9882
10	1.00E-04	-411.500	0.50237	0.75391	0.99549	-420.508	0.5028	0.8007	0.9792
11	1.00E-04	-434.784	0.50247	0.75867	0.99582	-445.956	0.5020	0.7866	0.9748
12	1.00E-04	-458.962	0.50263	0.77622	0.99645	-488.104	0.5016	0.7789	0.9724
13	1.00E-04	-476.254	0.50116	0.69237	0.99208	-553.635	0.4939	0.4248	0.9908

Таблица 2. Результаты метрик тестовой и валидационной выборок после обучения на U-Net++L3.

epoch	learning_rate	loss	mean_iou_u	precision	recall	val_loss	val_mean_iou_u	val_precision	val_recall
0	1.00E-04	-67,7703	0,50278	0,10310	0,95837	5,9693	0,49999	0,44572	0,14159
1	1.00E-04	-88,5164	0,50002	0,18656	0,99283	4,5429	0,49999	0,24182	0,13617
2	1.00E-04	-90,4698	0,49871	0,21907	0,99572	-22,0816	0,49999	0,52713	0,56461
3	1.00E-04	-91,1579	0,49797	0,25335	0,99715	-62,1069	0,49999	0,57415	0,82788
4	1.00E-04	-89,5654	0,49919	0,22374	0,99367	-78,3743	0,49998	0,09330	0,98643

Результаты метрик, собранных во время обучения модели U-Net++L3 на 5 эпохах представлены в таблице 2.

Для наглядного сравнения каждой из метрик по отдельности в программе Excel были построены соответствующие диаграммы.

По результатам метрики Mean IoU [15], значения для тестовых и валидационной выборок у каждой модели были близки к 0.5. Это означает, что отношение пересечения маски и оригинального изображения к их объединению примерно одинаковое. На рисунке 19 представлены изменения метрики Mean IoU для тренировочной и валидационной выборок во время обучения.

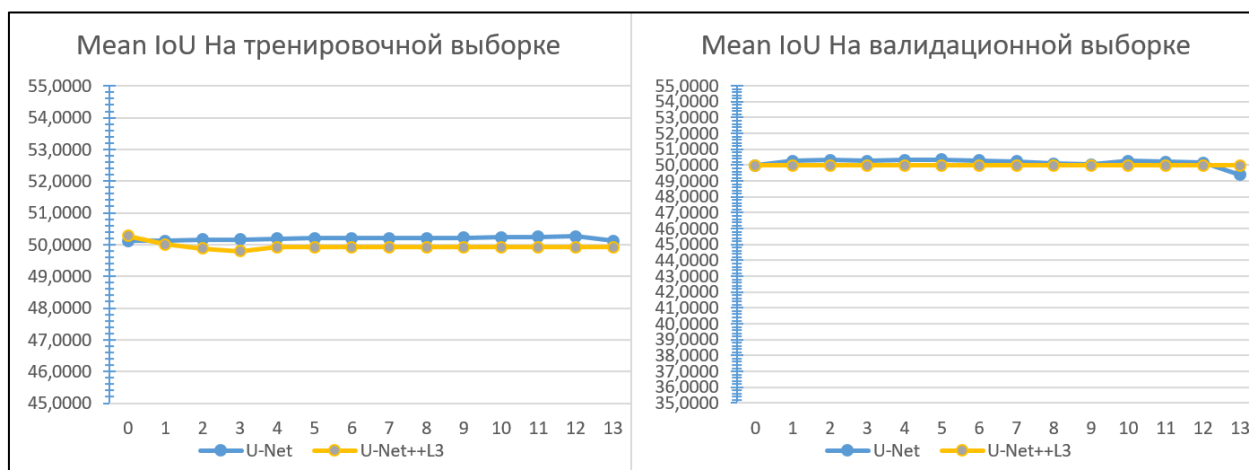


Рисунок 19. *Диаграмма с изменяющейся метрикой MeanIoU для тренировочной и валидационной выборок.*

Метрика «точность» [16] на тестовой выборке, у U-Net модели к концу обучения приблизилась к 0.9. У U-Net++L3 метрика осталась на уровне 0,2. На валидационной выборке результаты похожие. Эта метрика показывает на процент пикселей модель посчитала принадлежащими мыши на картинке и которые оказались верным предсказанием. Лучшие значения оказались у модели U-Net. На рисунке 20 представлены изменения метрики Precision для тренировочной и валидационной выборок во время обучения.

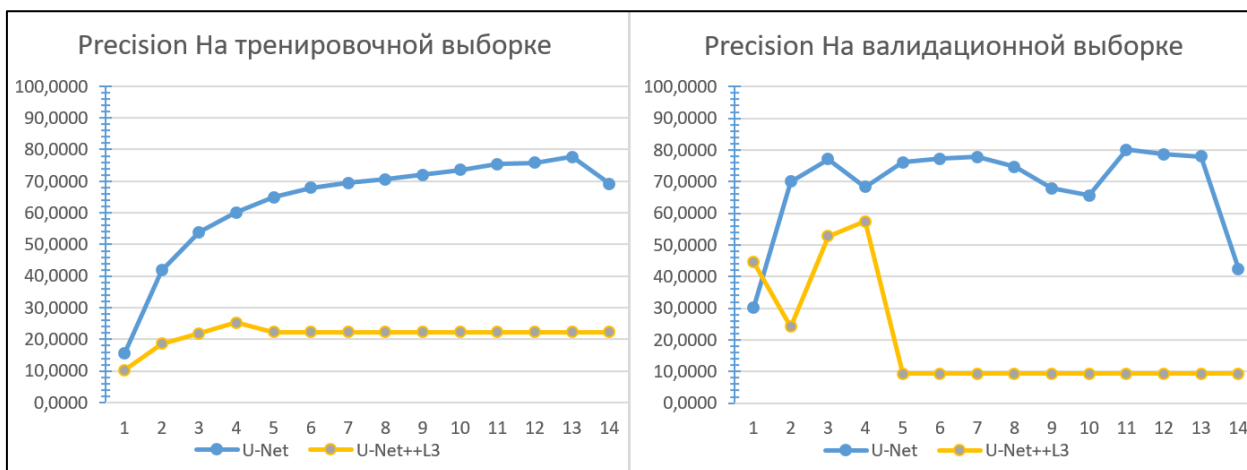


Рисунок 20. Диаграмма с изменяющейся метрикой *Precision* для тренировочной и валидационной выборок.

Метрика «полнота» [16] показывает насколько полно модель различает мышь на фоне. Для тестовой и валидационной выборок, метрика Recall к концу обучения доходила до 99%-100%. Лучшие значения оказались у модели на U-Net. На рисунке 21 представлены изменения метрики Recall для тренировочной и валидационной выборок во время обучения.

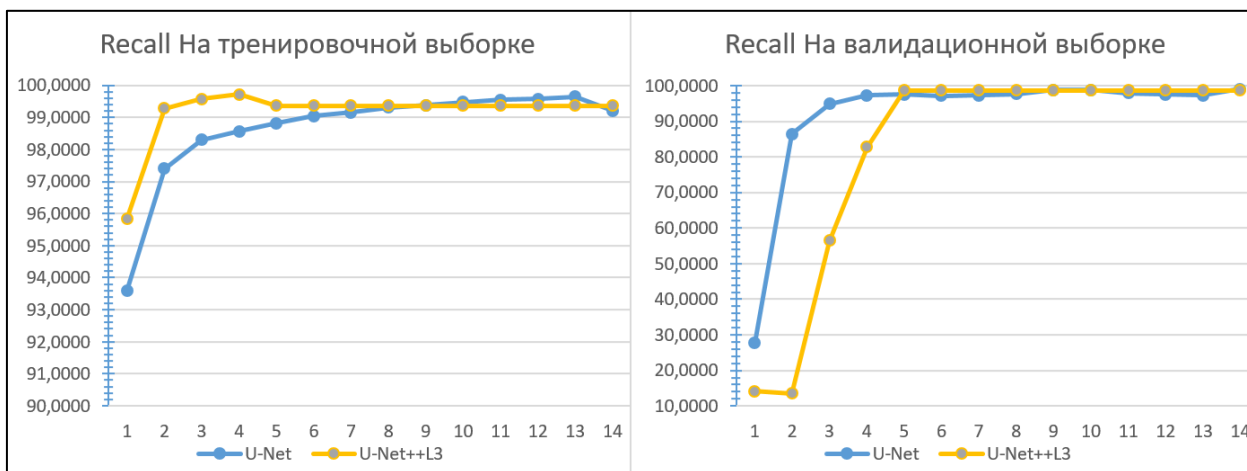


Рисунок 21. Диаграмма с изменяющейся метрикой *Recall* для тренировочной и валидационной выборок.

По метрике «переобучения» [17], на тестовой и валидационной выборках лучше всего показала себя модель U-Net++L3. К концу обучения у модели U-Net начиналось заметное переобучение, когда у архитектуры на основе U-Net++, значение не превышало -200. На рисунке 22 представлены

изменения метрики Loss для тренировочной и валидационной выборок во время обучения.

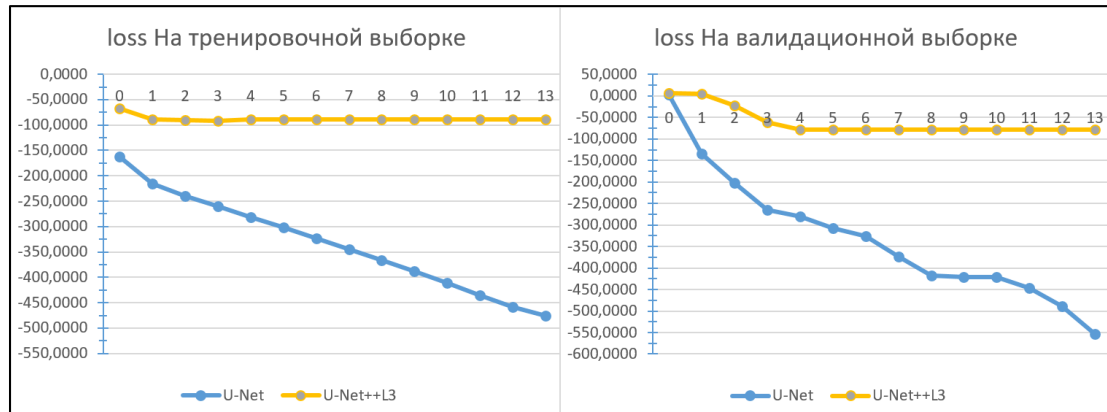


Рисунок 22. *Диаграмма с изменяющейся метрикой Loss для тренировочной и валидационной выборок.*

По полученным результатам, лучшей архитектурой для сегментации оказалась и U-Net++3.

2.2.2. Результаты сегментации

По результатам сегментации можно заметить, что модель не смогла точно распознать силуэт мыши на картинке. На полученной маске присутствует много шумов как на полу, так и на стенках. Результат сегментации на модели U-Net на 14 эпохах обучения представлен на рисунке 23.

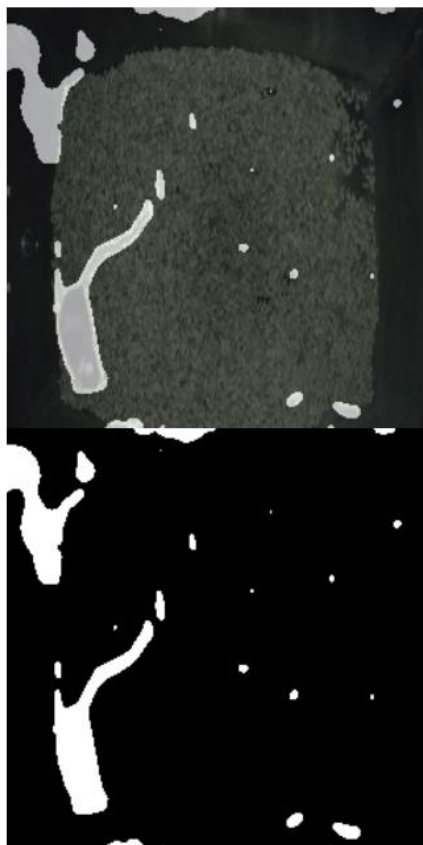


Рисунок 23. *Результат сегментации модели на архитектуре U-Net с 14 эпохами обучения.*

Результат сегментации U-Net субъективно неудовлетворительный.

Далее рассмотрим сегментацию на модели U-Net++L3. После десяти эпох обучения, на полученной маске, при наличии минимальных шумов, четко видны части мыши включая ее хвост. Результат сегментации на модели U-Net++L3 на 10 эпохах обучения представлен на рисунке 24.

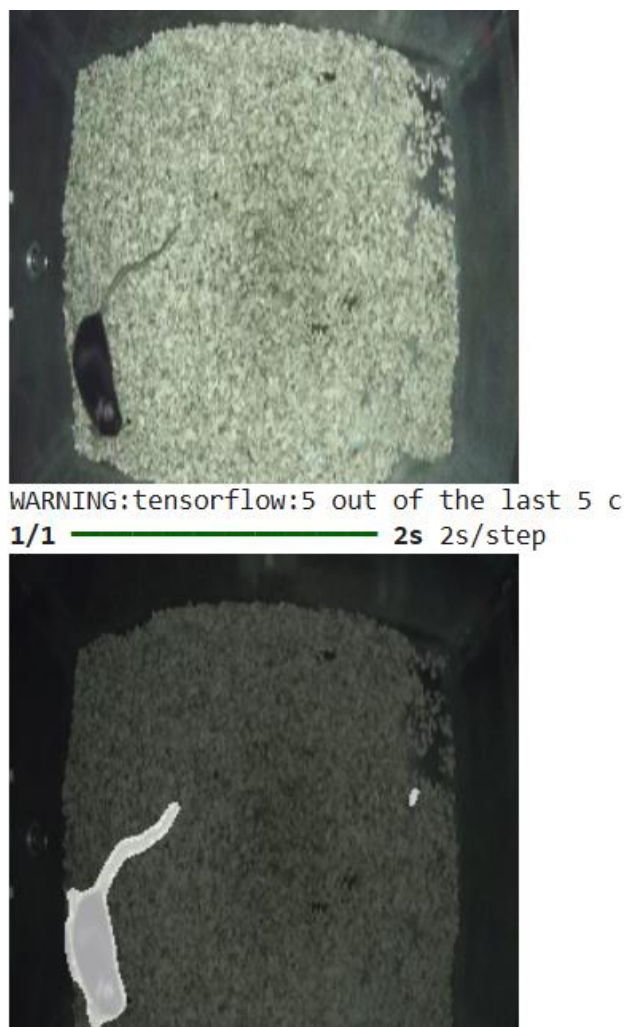


Рисунок 24. *Результат сегментации модели на архитектуре U-Net++L3 с 10 эпохами обучения.*

Так-как на 10 эпохах, у силуэта мыши появились странные выпуклости в районе хвоста и мордочки предположительно из-за переобучения, было решено сократить количество эпох обучения до 5, тем самым улучшив предыдущий результат. Сегментация на модели U-Net++L3 на 5 эпохах обучения представлена на рисунке 25.



Рисунок 25. *Результат сегментации модели на архитектуре U-Net++L3 с 5 эпохами обучения.*

2.2.3. Выбор лучшей модели

По результатам обучения и выборочной сегментации, лучше всего проявила себя модель с архитектурой U-Net++L3 на 5 эпохах обучения.

2.3. Использование YOLO для сегментации изображений

В процессе обучения, YOLO использовалась для решения задач по детекции. В рамках текущей практики необходимо было создать программу, способную определять, к какому объекту принадлежит каждый пиксель на видео, выделяя животное одним цветом, а фон — другим.

Датасет для обучения используется тот-же, что и для предыдущей работы, но с ограничением в 105 картинок. Для создания тренировочных

данных использовался сервис Roboflow. На рисунке 26 представлен процесс создания датасета.

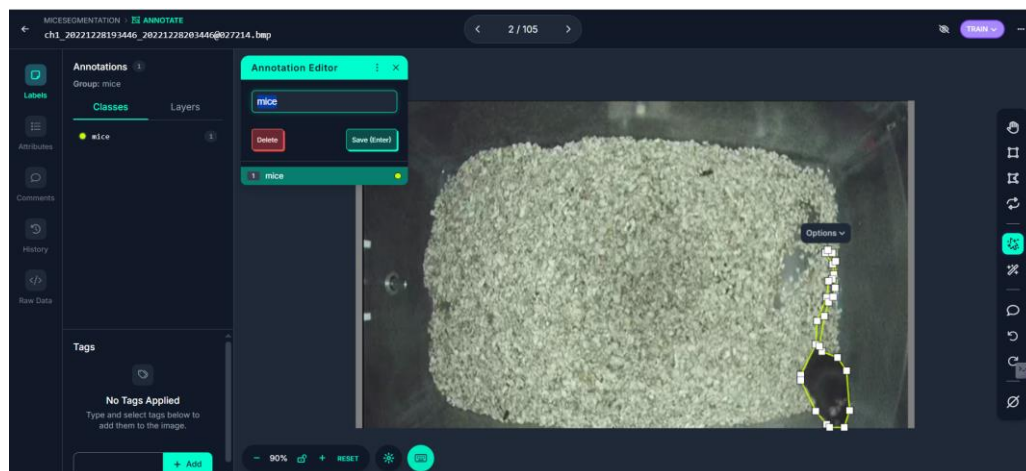


Рисунок 26. Процесс создания датасета.

На шаге аугментации для картинок были сгенерированы новые вариации обучающих данных, за счет вращения изображения на 15 градусов и добавления шума. В результате датасет содержал в себе 282 картинки, из которых 88% - обучающие данные, 8% - валидационные, 4% - тестовые. На рисунке 27 представлен результат формирования тренировочного датасета.

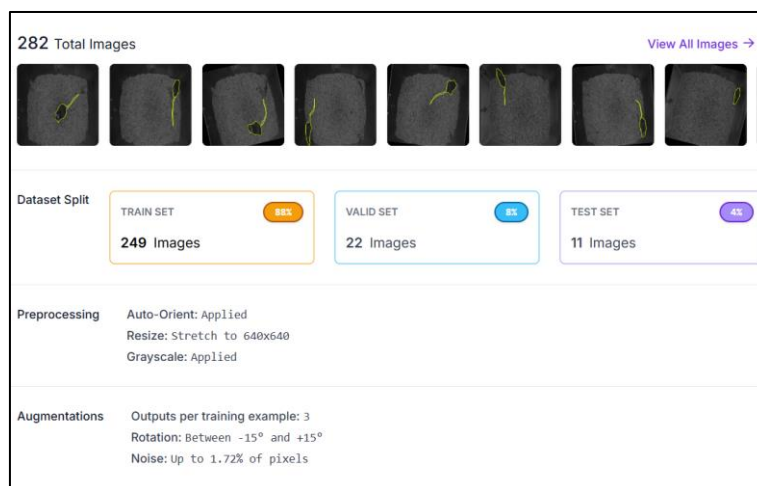


Рисунок 27. Результат формирования тренировочного датасета.

Для обучения модели из библиотеки ultralytics была скачана модель yolo11n-seg.pt для сегментации, которой характерна высокая скорость предсказания. Сформированный ранее датасет импортируется с помощью кода представленного на рисунке 28.

```
#import
from roboflow import Roboflow
rf = Roboflow(api_key="wGlZ3vOrEX0zIGqbJUr0")
project = rf.workspace("yolo-zad4").project("micesegmentation")
version = project.version(2)
dataset = version.download("yolov11")
```

Рисунок 28. Код, для импорта сформированного на Roboflow датасета.

Было проведено два обучения на 10 и 16 эпохах. Обучение модели на 10 эпохах, составило 3 минуты, на 16 эпохах – 4.5 минуты. По результатам точности обучение на 16 эпохах показывает лучший результат. Результаты обучения на валидационных данных приведены на рисунке 29.

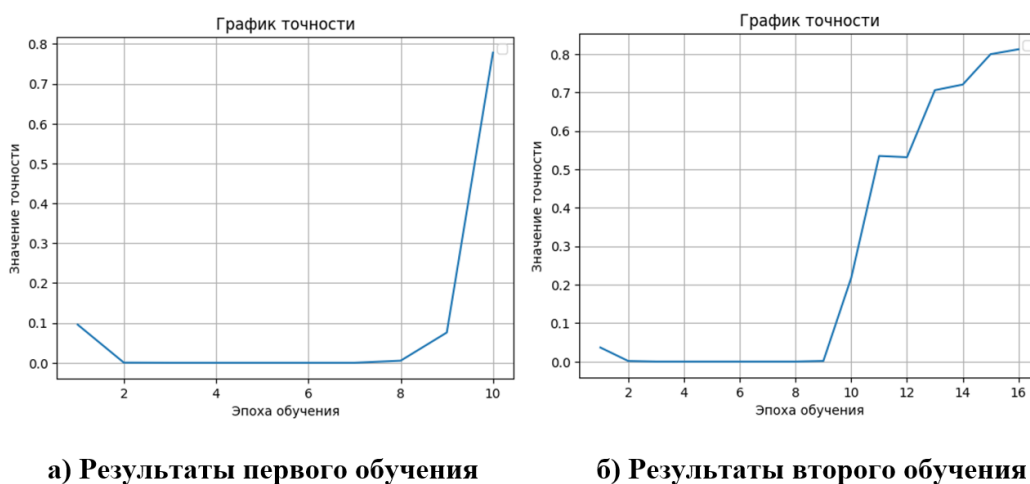


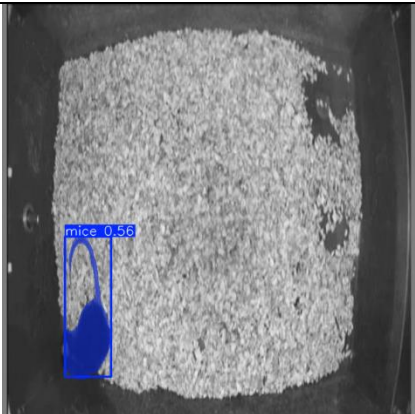
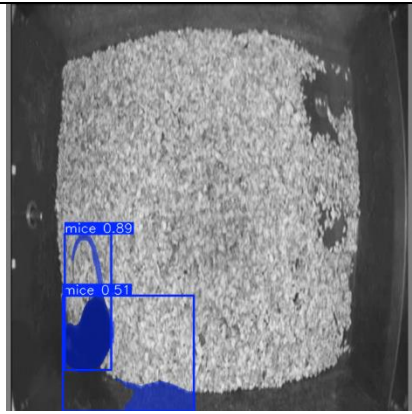
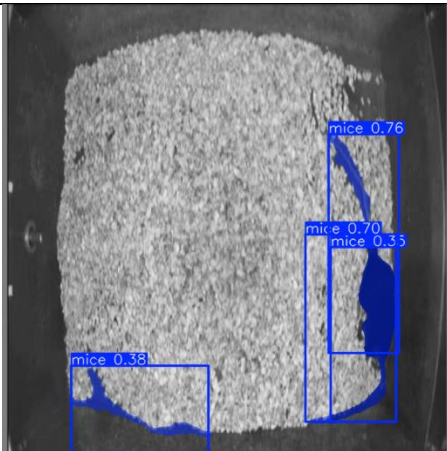
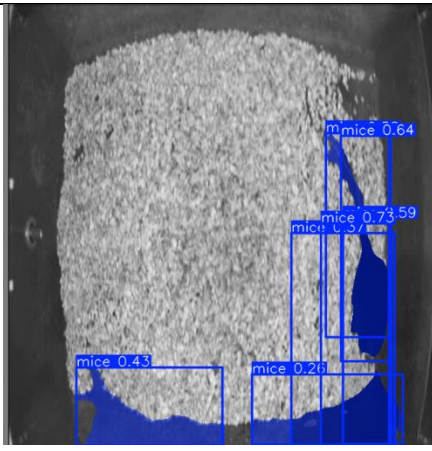
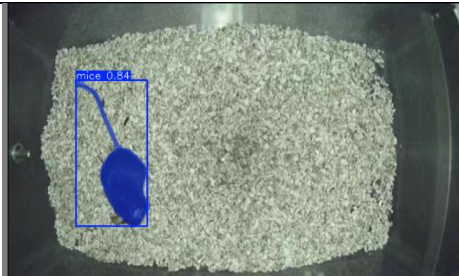
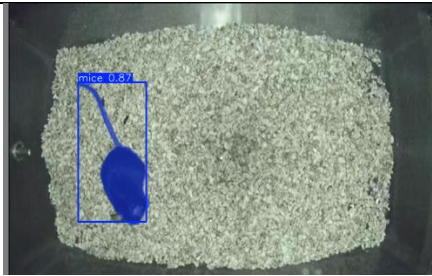
Рисунок 29. Точность моделей на валидационных данных.

Следующим шагом была сегментация тестовых данных датасета. Первая модель, обученная на 10 эпохах, правильно нашла всех мышей на фото, хотя на некоторых изображениях, просвечивающий пол клетки тоже считался за мышь, из-за чего у на полученной маске можно было обнаружить объект с двумя хвостами.

Для решения этой проблемы, модель обучили второй раз уже на 16 эпохах. Модель правильно и более уверенно нашла всех мышей на фото и двуххвостых мышей на предсказанных картинках не оказалось, однако теперь к просвечивающемуся полу так же добавились темные стенки клетки, которые модель тоже считала за объект.

Результаты сегментации изображений на тестовых выборках представлены в таблице 3.

Таблица 3. Результаты сегментации изображений на двух моделях.

	Модель обученная на 10 эпохах	Модель обученная на 16 эпохах
На тестовых данных из датасета		
		
На изображениях, не участвовавших в обучении		

Таким образом, хотя точность сегментирования мыши увеличилась, увеличением эпох не удалось решить проблему с просвечивающим полом и стенками. Возможным решением будет увеличение тренировочных данных или подбора лучшей модели.

3. Результаты и выводы по проделанной работе

В ходе практической работы в Google Collab была разработана программа для сегментации изображений с использованием архитектуры U-Net++ и проведены опыты по сегментации, по окончании которых сравнивались результаты сегментации, полученные с помощью U-Net++ и U-Net. Эти эксперименты позволили оценить, какая из моделей демонстрирует наилучшие результаты в задаче сегментации, а также выявить сильные и слабые стороны каждой из архитектур.

Кроме того, была разработана программа, использующая YOLO в связке с платформой Roboflow, для определения принадлежности каждого пикселя к определенному классу объектов, в данном случае — к мыши на изображениях. Этот подход дал возможность проводить точный анализ объектов на изображениях, но в то же время, с точки зрения алгоритмической сложности и задач, он больше относится к области аналитики и подготовки данных для обучения, чем к задаче детальной сегментации изображений.

Лично мне ближе работа с задачами сегментации, так как она требует более глубокого понимания методов для формирования различных архитектур и позволяет добиться большей точности в выделении объектов. Однако использование сервиса Roboflow с YOLO может дать дополнительные возможности в будущих проектах, особенно в тех случаях, когда требуется быстрая и эффективная обработка больших объемов данных или быстрое и удобное формирование датасетов.

ЗАКЛЮЧЕНИЕ

В результате выполнения научно исследовательской работы были закреплены теоретические знания, полученные при изучении дисциплин ранее в этом году. Приобретен опыт в решении реальных профессиональных задач, в рамках проделанной работы стояла задача сегментации изображения разными способами. В заключении полученные опыт был проанализирован и по полученным результатам определено направление для моего развития в следующем году.

Были реализованы задачи, а именно:

- закреплен материал дисциплин, изученных ранее в этом году,
- изучены методики сегментации изображений способами компьютерного зрения,
- разработаны и обучены модели для сегментации изображений на основе искусственных нейронных сетей, существующих архитектур на примере U-Net и YOLO,
- описано интересное направление ИИ, для написания выпускной квалификационной работы.

Таким образом, цель и поставленные задачи были достигнуты.

Для дальнейшего развития необходимо:

- выбрать тему выпускной работы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Open Source Computer Vision [Электронный ресурс] // docs.opencv.org [Сайт]. URL: <https://docs.opencv.org/3.4/index.html> (дата последнего обращения 10.06.2025) или <https://opencv.org/> (дата последнего обращения 15.06.2025).
- 2 В. А. Офицеров, А. С. Конушин. Нейросетевые методы сегментации изображений высокого разрешения [Научная статья] // cyberleninka.ru [Сайт]. URL: <https://cyberleninka.ru/article/n/neyrosetevye-metody-segmentatsii-izobrazheniy-vysokogo-razresheniya/viewer> (дата последнего обращения 10.06.2025).
- 3 Liang-Chieh Chen, George Papandreou, Florian Schroff, Hartwig Adam. Rethinking Atrous Convolution for Semantic Image Segmentation [Научная статья] // arxiv.org [Сайт]. URL: <https://arxiv.org/pdf/1706.05587> (дата последнего обращения 10.06.2025).
- 4 Kaiming He, Georgia Gkioxari, Piotr Dollar, Ross Girshick. Mask R-CNN, 2018 [Научная статья] // arxiv.org [Сайт]. URL: <https://arxiv.org/pdf/1703.06870> (дата последнего обращения 10.06.2025).
- 5 Abirami Vina. What is R-CNN? A quick overview [Электронный ресурс] // www.ultralitics.com [Сайт]. URL: <https://www.ultralitics.com/ru/blog/what-is-r-cnn-a-quick-overview> (дата последнего обращения 10.06.2025).
- 6 Vijay Badrinarayanan, Alex Kendall, SegNet: A Deep Convolutional EncoderDecoder Architecture for Image Segmentation, 2016 [Научная статья] // arxiv.org [Сайт]. URL: <https://arxiv.org/pdf/1511.00561> (дата последнего обращения 10.06.2025).
- 7 Визуальные трансформеры (ViT) [Электронный ресурс] // habr.com [Сайт]. URL: <https://habr.com/ru/companies/otus/articles/849756/> (дата последнего обращения 10.06.2025).
- 8 Olaf Ronneberger, Philipp Fischer, U-Net: Convolutional Networks for Biomedical Image Segmentation, 2015. [Научная статья] // arxiv.org [Сайт].

- URL: <https://arxiv.org/pdf/1505.04597> (дата последнего обращения 10.06.2025).
- 9 U-Net: нейросеть для сегментации изображений [Электронный ресурс] // neurohive.io [Сайт]. URL: <https://neurohive.io/ru/vidy-nejrosetej/u-net-image-segmentation/> (дата последнего обращения 10.06.2025).
- 10 Unet++ Architecture Explained [Электронный ресурс] // www.geeksforgeeks.org [Сайт]. URL: <https://www.geeksforgeeks.org/machine-learning/unet-architecture-explained/> (дата последнего обращения 10.06.2025).
- 11 Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. UNet++: A Nested U-Net Architecture for Medical Image Segmentation. [Научная статья] // arxiv.org [Сайт]. URL: <https://arxiv.org/pdf/1807.10165v1> (дата последнего обращения 10.06.2025).
- 12 Computer Vision Tasks supported by Ultralytics YOLO11. [документация] // docs.ultralytics.com [Сайт]. URL: <https://docs.ultralytics.com/ru/tasks/> (дата последнего обращения 15.06.2025).
- 13 Maxim Dupont. History of YOLO: From YOLOv1 to YOLOv10. [Электронная статья] // [labelvisor.com](https://www.labelvisor.com) [Сайт]. URL: <https://www.labelvisor.com/history-of-yolo-from-yolov1-to-yolov10/> (дата последнего обращения 15.06.2025).
- 14 Google Colab вместо Jupyter Notebook: плюсы и особенности работы для новичков [Электронный ресурс] // habr.com [Сайт]. URL: https://habr.com/ru/companies/yandex_praktikum/articles/825754/ (дата последнего обращения 10.06.2025).
- 15 MeanIoU. [документация] // haibal.com [Сайт]. URL: <https://haibal.com/documentation/metric-mean-iou/> (дата последнего обращения 10.06.2025).
- 16 Метрики классификации и регрессии. [Электронный ресурс] // education.yandex.ru [Сайт]. URL: education.yandex.ru

<https://education.yandex.ru/handbook/ml/article/metriki-klassifikacii-i-regressii> дата последнего обращения 10.06.2025).

- 17 Shruti Jadon. A survey of loss functions for semantic segmentation, 2020 [Научная статья] // arxiv.org [Сайт]. URL: <https://arxiv.org/pdf/2006.14822> (дата последнего обращения 10.06.2025).

Приложение А

ссылка на репозиторий с кодом программы и результатами сегментации

Репозиторий проекта можно найти, перейдя по ссылке:

<https://github.com/Uranus28/proektnoPraktika25> (github.com)

**Рекомендации и замечания руководителя практики в период
прохождения студентом практики**

За период преддипломной практики студент изучил Им
успешно/неуспешно освоены, выполнен (выполнен не
полностью, ...)

Результаты преддипломной практики оформлены в виде подробного
отчета, к которому имеются достаточные наглядные приложения (при
наличии).

Цель и поставленное задание преддипломной практики выполнены
полностью/ не выполнены. Студент собрал достаточный/ недостаточный
практический материал для написания дипломной работы.

Рекомендованная оценка ...

ФИО руководителя