**Министерство науки и высшего образования Российской Федерации**

**ФГАОУ ВО «УрФУ имени первого Президента России Б.Н. Ельцина»**

Кафедра «школа бакалавриата (школа)»

СЕГМЕНТАЦИЯ И КЛАССИФИКАЦИЯ РЕНТГЕНОВСКИХ СНИМКОВ ЛЕГКИХ

Отчет по дисциплине «Теория принятия решений»

Студент Бордунов Александр Максимович

**ФИО студента**

Специальность (направление подготовки) 09.03.04 Программная инженерия

Группа РИ-220943

Екатеринбург 2024

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc173763693)

[1 Теоретическая часть 5](#_Toc173763694)

[2 Практическая часть 7](#_Toc173763695)

[2.1 Подготовка данных для обучения 7](#_Toc173763696)

[2.2 Обучение модели 15](#_Toc173763697)

[ЗАКЛЮЧЕНИЕ 18](#_Toc173763698)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 19](#_Toc173763699)

# **ВВЕДЕНИЕ**

Компьютерное зрение — это область искусственного интеллекта, которая занимается разработкой алгоритмов и систем для анализа и понимания цифровых изображений и видео. Одним из ключевых направлений здесь является классификация изображений. Эта задача заключается в определении категории или класса, к которому относится данное изображение. Как отмечает Ян Лекун, "классификация изображений — фундаментальная проблема в компьютерном зрении с множеством практических приложений".[2]

Процесс классификации включает несколько этапов: предварительную обработку изображения (например, нормализация размера и цветовая коррекция), извлечение признаков (таких как текстура, форма и цвет), и применение алгоритмов машинного обучения, обученных на размеченных данных. Фейсал Рашид подчеркивает, что "качество извлечённых признаков критически важно для точности классификации". [1]

С внедрением глубоких нейронных сетей, таких как свёрточные нейронные сети (CNN), удалось значительно повысить точность классификации, иногда превосходя человеческие возможности. Яннис Амаратунга утверждает, что "глубокие нейронные сети показали выдающиеся результаты в этой области". Дополнительно, методы увеличения данных и регуляризация помогают улучшить обобщающую способность моделей, избегая переобучения.

Классификация изображений находит применение в различных сферах, таких как распознавание объектов, медицинская диагностика, и агропромышленный комплекс. В распознавании объектов технологии используются в автономных транспортных средствах и системах безопасности. В медицине автоматизация анализа снимков помогает в диагностике. В агропромышленном комплексе анализ изображений используется для мониторинга состояния культур и выявления болезней. Как утверждает Сюзан Чжан, "компьютерное зрение и классификация изображений открывают новые возможности для автоматизации задач, ранее выполнявшихся только людьми".

Цель практики заключалась в создании системы из 2-х моделей (классификации и сегментации) для автоматического определения заболевания легких и его выделения на рентгеновском снимке.

# **1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

Для создания программы классификации изображений используется PyTorch — библиотека для разработки приложений на основе глубокого обучения. PyTorch обладает рядом ключевых возможностей:

1. Гибкость при разработке решений благодаря возможности писать собственные функции и доступу к математическому аппарату. Это позволяет адаптировать и оптимизировать модели под конкретные задачи и данные.
2. Поддержка интерактивной отладки. Найти причину ошибки несложно, поскольку можно напрямую обратиться к исходному коду библиотеки, что значительно упрощает процесс разработки и тестирования.
3. Хорошая документация. PyTorch предоставляет подробное описание каждой функции с примерами, что облегчает изучение и использование библиотеки.
4. Обширное комьюнити разработчиков. На форумах, таких как StackOverflow, публикуются разборы популярных проблем, что помогает быстро находить решения и обмениваться опытом.
5. Простота адаптации проектов, в которых используется NumPy. Для интеграции достаточно добавить буквально несколько строк кода.

PyTorch широко используется в академических исследованиях и промышленных приложениях для решения задач компьютерного зрения, обработки естественного языка, генеративных моделей и многого другого. Важной особенностью PyTorch является его способность работать с динамическими вычислительными графами, что позволяет изменять архитектуру модели "на лету". Это особенно полезно при разработке сложных моделей, таких как рекуррентные нейронные сети и генеративные состязательные сети (GAN).

Дополнительно, PyTorch поддерживает интеграцию с другими популярными библиотеками и инструментами, такими как TensorBoard для визуализации и torchvision для работы с датасетами и предобученными моделями. Это делает PyTorch удобным инструментом как для исследований, так и для разработки[4]промышленных приложений.

PyTorch также активно развивается, и сообщество постоянно вносит улучшения и добавляет новые возможности, что делает библиотеку актуальной и востребованной в современных проектах по машинному обучению и глубокому обучению.

# **2 Практическая часть**

Выполнение работы проводилось в несколько этапов

## 2.1 Подготовка данных для обучения

Импортирование нужных библиотек (Рисунок 1)

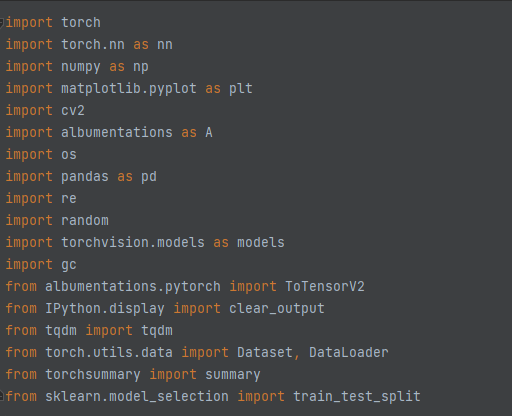


Рисунок 1 – Импорт нужных библиотек

train\_transforms — это переменная, которая представляет собой серию преобразований изображений, применяемых к набору данных.

Каждый шаг преобразования:

1. A.Resize((256, 256)): это преобразование изменяет размер входного изображения до фиксированного размера 256x256 пикселей. Изменение размера часто необходимо, чтобы убедиться, что все изображения в наборе данных имеют одинаковые размеры;

2. A.HorizontalFlip(p=0.3): это преобразование случайным образом отражает изображение по горизонтали с вероятностью 30%. Оно помогает увеличить разнообразие обучающих данных, создавая новые образцы, которые являются горизонтально отзеркаленными версиями исходных изображений;

3. transforms.Rotate(limit=45, p=0.3): это преобразование случайным образом поворачивает изображение на максимальный угол в 45 градусов. Подобно предыдущему преобразованию, оно вводит дополнительные вариации в обучающие данные, делая модель более устойчивой к различным ориентациям объектов на изображениях;

4. transforms.ToTensorV2(): это преобразование преобразует изображение из объекта PIL (Python Imaging Library) в тензор PyTorch. Тензоры PyTorch являются основной структурой данных, используемой для обучения моделей глубокого обучения. Этот шаг необходим, потому что большинство фреймворков глубокого обучения, включая PyTorch, ожидают, что входные данные будут в формате тензора;

Применяя эти преобразования к каждому изображению в наборе данных, можно улучшить разнообразие набора данных, стандартизировать формат ввода и улучшить способность модели извлекать значимые характеристики из данных. (Рисунок 2)

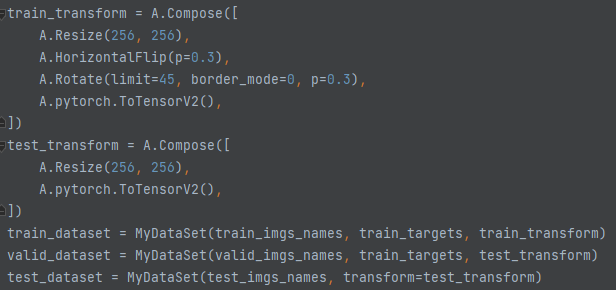


Рисунок 2 – Преобразование изображения

Определение конфигурационного класса (Рисунок 3)

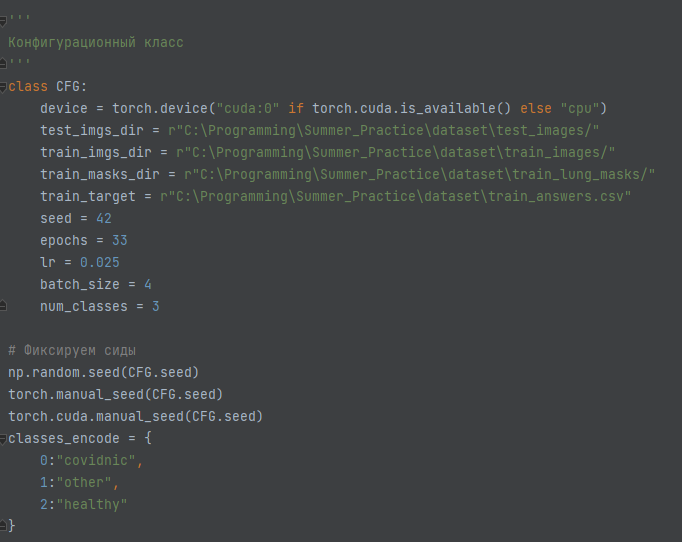


Рисунок 3 – Конфигурационный класс

train\_images\_dir является путем до папки, где находятся тренировочные данные

Код, выполняющий разделение набора данных на обучающий, валидационный и тестовый наборы (Рисунок 4). По каждому шагу:

1. train\_size = int(0.8 \* len(image\_datasets)): здесь мы определяем размер обучающего набора данных. Мы берем 80% от общего количества изображений в image\_datasets для обучающего набора. Это число вычисляется как 80% от общего размера набора данных и округляется до целого числа;

2. val\_size = len(image\_datasets) – train\_size: Размер валидационного набора данных определяется как разница между общим размером набора данных и размером обучающего набора;

3. train\_dataset, val\_dataset = torch.utils.data.random\_split(image\_datasets, [train\_size, val\_size]): здесь происходит фактическое разделение набора данных. Функция torch.utils.data.random\_split принимает исходный набор данных image\_datasets и список с размерами для каждого нового набора данных (обучающего и валидационного). Она затем случайным образом разделяет исходный набор данных на две части в соответствии с указанными размерами.

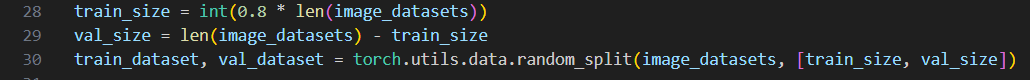


Рисунок 4 – Разделение набора данных на обучающий и валидационный наборы

Код, создающий три загрузчика данных: train\_loader, valid\_loader и test\_dataloader (Рисунок 5). Загрузчики данных используются для загрузки данных в модель пакетами (batch) во время обучения и валидации.

DataLoader — это класс PyTorch, который позволяет загружать данные из набора данных в пакетах (batch) и предоставляет удобный интерфейс для итерации по данным во время обучения модели.

Train\_loader и val\_loader создаются на основе train\_dataset и val\_dataset соответственно. Параметр batch\_size указывает, сколько изображений будет загружаться в каждый пакет (batch). В данном случае мы загружаем 32 изображения в каждый пакет.

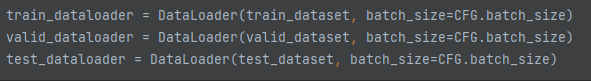


Рисунок 5 – Код, создающий два загрузчика данных

Пример тренировочных данных (Рисунок 6)

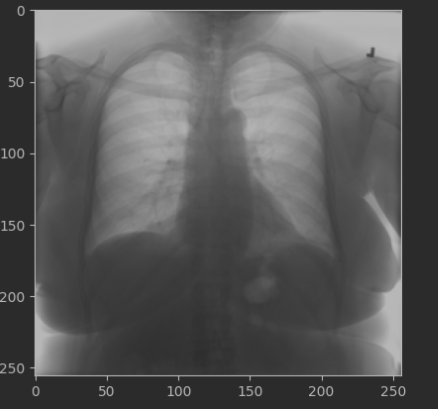


Рисунок 6 – Снимок легких их тренировочного набора данных

Класс `MyDataSet` (Рисунок 7), основанный на PyTorch `Dataset`, загружает изображения и метки для моделей глубокого обучения.

- Инициализация: принимает имена изображений, метки и трансформации, определяет режим (обучение или тестирование), создаёт списки путей к изображениям и меткам.

- Загрузка данных: для каждого имени изображения создаёт путь и добавляет его в `imgs\_paths`. В режиме обучения добавляет пути к маскам и one-hot encoded метки.

- Метод `\_\_len\_\_`: возвращает количество изображений.

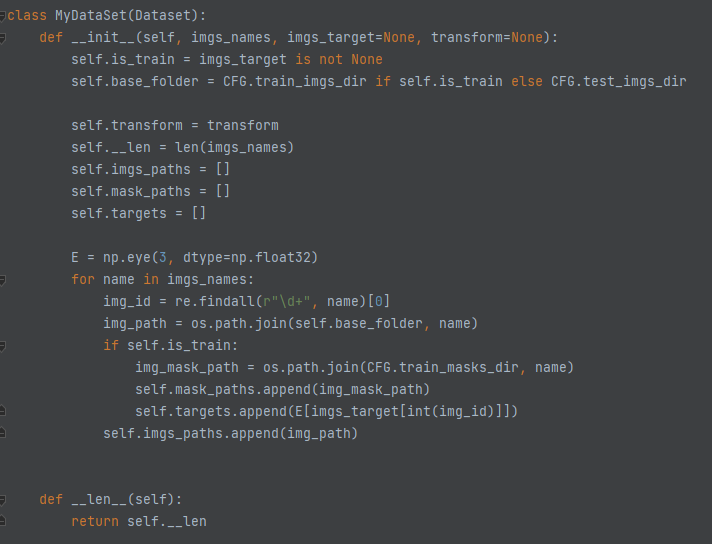


Рисунок 7 – Кастомный класс датасета

Этот класс упрощает загрузку и обработку данных в PyTorch.

Для решения задачи было решено использовать не одну, а две модели, объединение работы которых, позволило получить результаты гораздо лучше, чем в случае с одной моделью классификации. На рисунке 8 загружается не предобученная модель для сегментации unet.



Рисунок 8 – Загрузка модели сегментации

Рисунок 9 показывает загрузку архитектуры для классификации и замены последнего слоя классификатора для нашей задачи. Также происходит назначение модели на устройство (в нашем случае GPU)

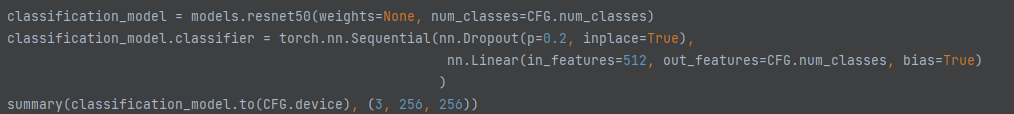


Рисунок 9 – Модель классификации и ее настройка

В итоге получилась составная модель (Рисунок 10)

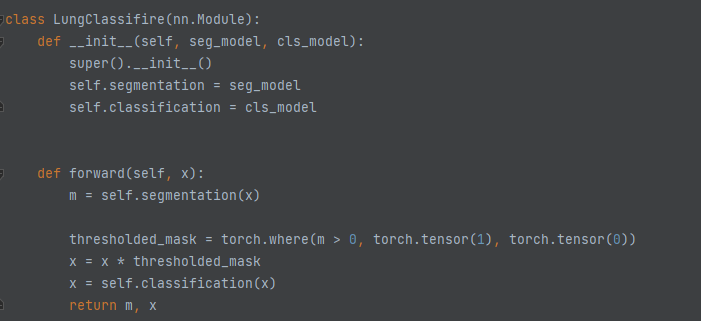


Рисунок 10 – Составная модель

## 2.2 Обучение модели

Программа обучает модель на протяжении 15 эпох (Рисунок 11). На каждой эпохе происходит тренировка модели и ее тестирование. Корректировка весов рассчитывается за счет функции вычисления обратного распространения ошибки. На каждой эпохе оценивается точность текущей модели и сохраняется лучший результат за все пройденные эпохи (Рисунок 12-13), который будет использован для реальных, тестовых данных после обучения.

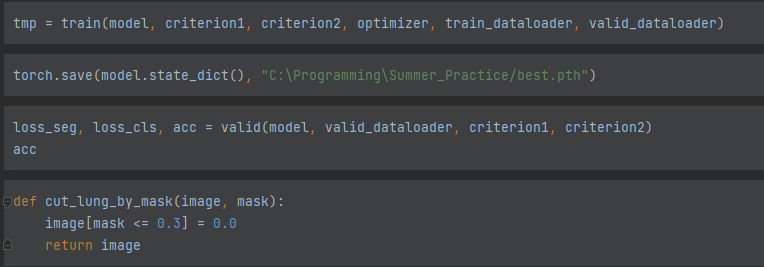


Рисунок 11 – Обучение модели и сохранение метрик

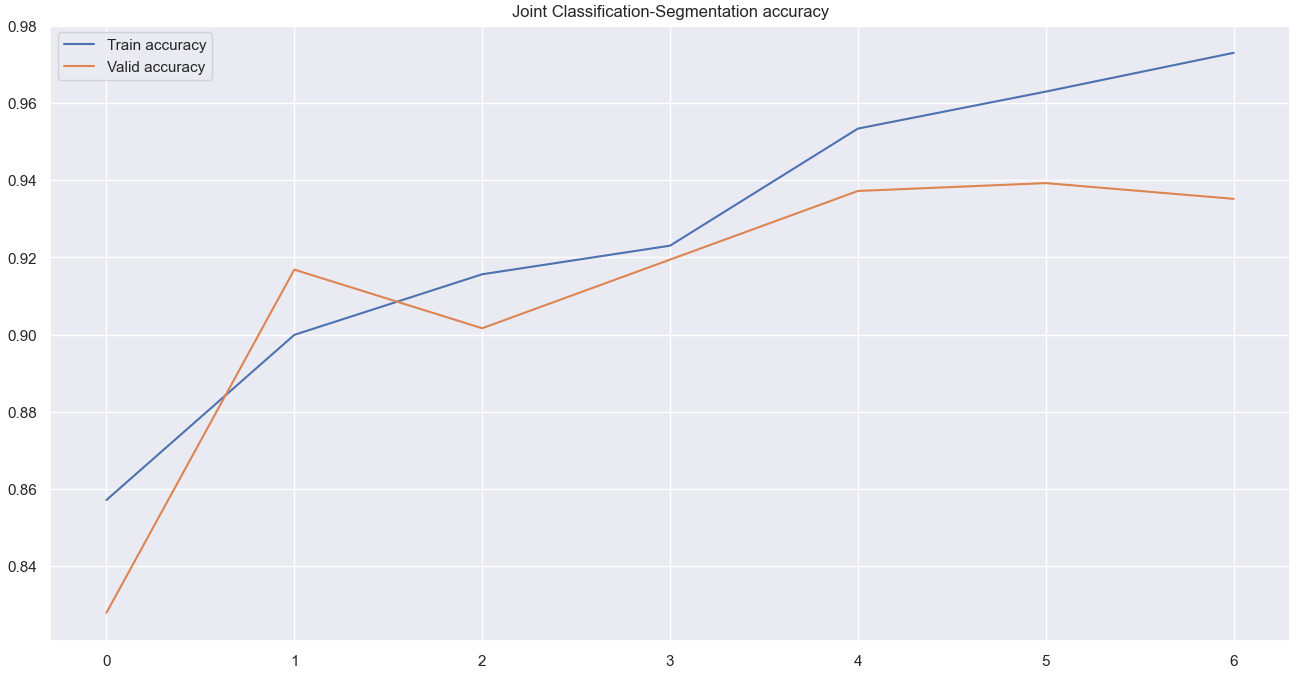


Рисунок 12 – график accuracy модели

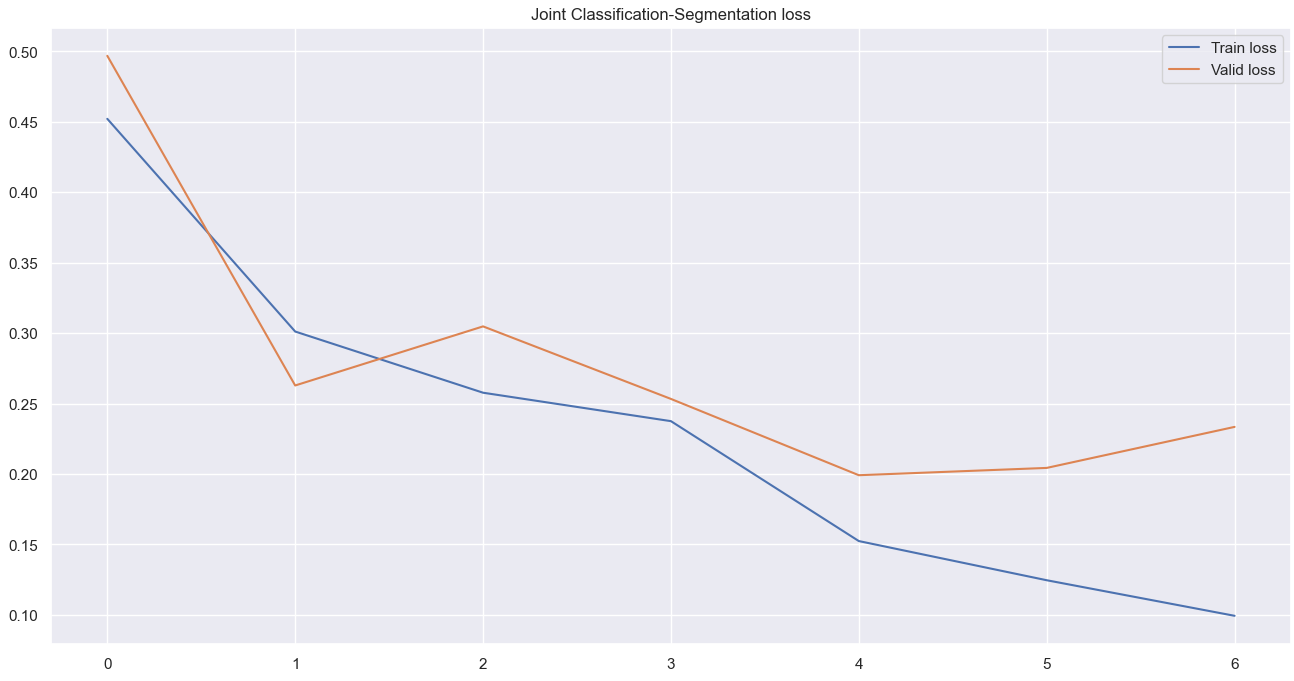


Рисунок 13 – график потерь модели

В конце программы для обучения находится код, оценивающий полученную обученную модель на тестовых данных (Рисунок 14)

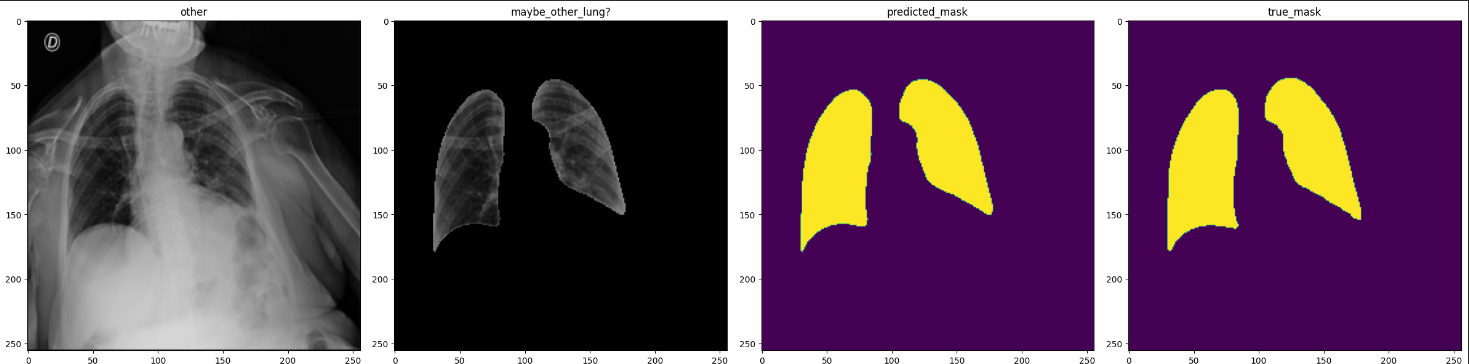


Рисунок 14 – Пример работы предсказателя

# **ЗАКЛЮЧЕНИЕ**

В данной работе были разработаны и протестированы две модели машинного обучения для задач классификации и сегментации на основе набора данных со снимками лёгких. Первая модель успешно классифицировала снимки на категории, такие как здоровые и с патологией, продемонстрировав высокую точность и надёжность. Вторая модель выполняла задачу сегментации, выделяя области лёгких и поражённые участки, что существенно облегчает процесс диагностики и анализа.

Результаты экспериментов показали, что обе модели способны эффективно обрабатывать медицинские изображения, предоставляя точные и информативные выводы. Модель классификации достигла высоких показателей точности, чувствительности и специфичности, что подтверждает её пригодность для использования в клинической практике. Модель сегментации продемонстрировала хорошую способность выделять поражённые области на снимках, что может быть полезно для последующего анализа и принятия решений врачами.

Применение этих моделей в реальной клинической практике может значительно ускорить процесс диагностики заболеваний лёгких, повысить точность постановки диагнозов и улучшить качество медицинской помощи. В дальнейшем возможно дальнейшее улучшение моделей, расширение набора данных и использование дополнительных методов обработки и анализа изображений для повышения эффективности и надёжности решений на основе машинного обучения.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. PyTorch – ваш новый фреймворк глубокого обучения // Хабр: [сайт]. – URL: <https://habr.com/ru/articles/334380/> (дата обращения: 11.07.2024);
2. Transforming and augmenting images // PyTorch : [сайт]. – URL: <https://pytorch.org/vision/0.13/transforms.html> (дата обращения: 12.07.2024);
3. What is Image Classification? (Что такое классификация изображений?) // URL: <https://www.geeksforgeeks.org/what-is-image-classification/> (дата обращения: 04.07.2024);
4. Сверточные нейронные сети для компьютерного зрения [1.1] Классификация изображений: Data-driven подход, k-Nearest Neighbor, train/val/test разделение // URL: <https://digiratory.ru/1492> (дата обращения: 04.07.2024);
5. PyTorch: Learning Rate Schedules // CoderzColumn : [сайт]. – URL: <https://coderzcolumn.com/tutorials/artificial-intelligence/pytorch-learning-rate-schedules> (дата обращения: 14.07.2024)
6. Building a Convolutional Neural Network using PyTorch // GeeksForGeeks : [сайт]. – URL: <https://www.geeksforgeeks.org/building-a-convolutional-neural-network-using-pytorch/> (дата обращения: 13.07.2024)
7. С.И. Николенко. Глубокое обучение. Погружение в мир нейронных сетей. - Москва: ДМК Пресс, 2018. - 384 с;
8. ML: Введение в PyTorch: 3. Нейронные сети // URL: <https://qudata.com/ml/ru/NN_Base_Torch_NN.html> (дата обращения: 08.07.2024);
9. Рецепт обучения нейросетей // Хабр : [сайт]. – URL: <https://habr.com/ru/articles/541136/> (дата обращения: 13.07.2024)
10. Cross Entropy Loss: Intro, Applications, Code // URL: <https://www.v7labs.com/blog/cross-entropy-loss-guide> (дата обращения: 10.07.2024);
11. Resnet in PyTorch // URL: <https://www.scaler.com/topics/pytorch/pytorch-resnet/> (дата обращения: 10.07.2024);
12. Implementation of a CNN based Image Classifier using PyTorch:  
    <https://www.geeksforgeeks.org/implementation-of-a-cnn-based-image-classifier-using-pytorch/> (дата обращения: 14.07.2024);