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

ДОПУСТИТЬ К ЗАЩИТЕ

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

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Выявление аномалий на рентгеновских снимках легких с использованием нейросетевых технологий

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.04.2024.308-365.ВКР

Научный руководитель,
доцент кафедры СП, к.п.н.
_____ О.Н. Иванова

Автор работы,
студент группы КЭ-404
_____ Е.А. Светличная

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-404
Светличной Екатерине Алексеевне,
обучающейся по направлению
09.03.04 «Программная инженерия»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)
Выявление аномалий на рентгеновских снимках легких с использованием нейросетевых технологий.
2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.
3. **Исходные данные к работе**

3.1. Вьюгин В.В. Элементы математической теории машинного обучения: учебное пособие. – М.: МФТИ: ИППИРАН, 2010. – 231 с.

3.2. Python. [Электронный ресурс] URL: https://www.python.org/ (дата обращения: 09.03.2024 г.).

3.3. Тёрк М., Дэвис Р. Компьютерное зрение. Передовые методы и глубокое обучение. – М.: ДМК-Пресс, 2022. – 690 с.

4. **Перечень подлежащих разработке вопросов**

4.1. Произвести анализ предметной области.

4.2. Реализовать топологию искусственной нейронной сети, обучить ее и протестировать.

4.3. Разработать систему выявления аномалий на рентгеновских снимках.

4.4. Провести тестирование системы.

5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
доцент кафедры СП, к.п.н. О.Н. Иванова

Задание принял к исполнению Е.А. Светличная

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

На сегодняшний день в медицине существует огромный объем информации, которую необходимо изучать и анализировать, чтобы правильно поставить диагноз. Рентгеновские снимки являются наиболее распространенным методом исследования в медицине, но анализ большого количества снимков может быть очень сложным и требовать больших временных затрат. Использование нейросетевых технологий для выявления аномалий на рентгеновских снимках может значительно упростить и ускорить этот процесс. Данная система может помочь в раннем выявлении заболеваний. В свою очередь, это позволит сократить расходы на лечение и снизить количество ошибок в диагностике, что является важным вопросом в современной медицине.

Актуальность применения нейронных сетей для анализа рентгеновских снимков в современной медицине обусловлена необходимостью повышения эффективности и точности диагностики, а также сокращения временных и финансовых затрат. Внедрение таких технологий может стать важным шагом на пути к улучшению качества медицинского обслуживания.

Постановка задачи

Целью выпускной квалификационной работы является выявление аномалий на рентгеновских снимках легких с использованием нейросетевых технологий.

Для достижения поставленной цели необходимо решить следующие задачи:

1) произвести анализ предметной области;
2) реализовать выбранную топологию искусственной нейронной сети, обучить ее и протестировать;
3) разработать систему для выявления аномалий на рентгеновских снимках;
4) провести тестирование системы.

Структура и содержание работы

Работа состоит из введения, пяти глав, заключения и списка литературы. Объем работы составляет 49 страниц, объем списка литературы – 20 источников.

Первая глава «Анализ предметной области» описывает предметную область, в рамках которой выполняется данная работа. Она включает в себя анализ аналогичных проектов и анализ существующих решений для реализации проекта.

Вторая глава «Теоретическая часть» содержит теоретические сведения о виде искусственной нейронной сети, которая применяется в данной работе, а также описывает выбранный набор данных.

Третья глава «Проектирование приложения» содержит описание функциональных и нефункциональных требований к разрабатываемому приложению. Также в этом разделе представлены диаграммы, описывающие взаимодействие пользователя с разрабатываемой системой и общую архитектуру системы.

Четвертая глава «Практическая часть» описывает реализацию программируемой системы.

Пятая глава «Тестирование» содержит тестирование нейронной сети и пример результата работы нейронной сети.

В заключении прописаны результаты проделанной работы.

В приложении содержится код реализации нейронной сети.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Предметная область проекта

Рентгенография грудной клетки [18] является одним из наиболее распространенных рентгенографических исследований. Рентгенографическое исследование способствует выявлению патологических изменений в области грудной клетки. Затемнения, просветления, снижение, повышение прозрачности легочного фона свидетельствует о том, что обнаружена патология. С помощью рентгена грудной клетки можно диагностировать такие заболевания, как пневмония, туберкулез, тромбоэмболия легочной артерии, пневмоторакс, эхинококкоз, различные опухолевые заболевания легких, бронхов и многие другие.

Основными показаниями к назначению рентгенографии легких являются длительный кашель, кровь в мокроте, одышка, постоянные боли в грудной клетке, повышенная температура тела.

Использование нейросетевых технологий при анализе рентгенограмм грудной клетки может значительно упростить процесс постановки диагноза, минимизировать ошибки и повысить эффективность диагностики различных заболеваний. Рентгенография грудной клетки является незаменимым диагностическим методом, который позволяет выявлять широкий спектр патологических изменений в организме человека.

1.2. Сравнительный анализ аналогов

Data-Efficient Vision Transformers for Multi-Label Disease Classification on Chest Radiographs [1]

В данной статье говорится об использовании Vision Transformers (ViTs) вместо сверточных нейроных сетей для классификации заболеваний на рентгеновских снимках легких. Для оценки качества работы моделей используется набор данных CheXpert. ViT обычно требуют чрезмерного объема обучающих данных, что представляет собой препятствие в медицинской сфере, поскольку сбор больших наборов медицинских данных связан с

высокими затратами. В этой работе сравнивается эффективность классификации ViT и CNN для разных размеров наборов данных и оцениваются более эффективные варианты ViT (DeiT). Результаты (рисунок 1) показывают, что, хотя производительность между ViT и CNN находится на одном уровне с небольшим преимуществом для ViT, DeiT превосходит первые, если для обучения доступен достаточно большой набор данных.

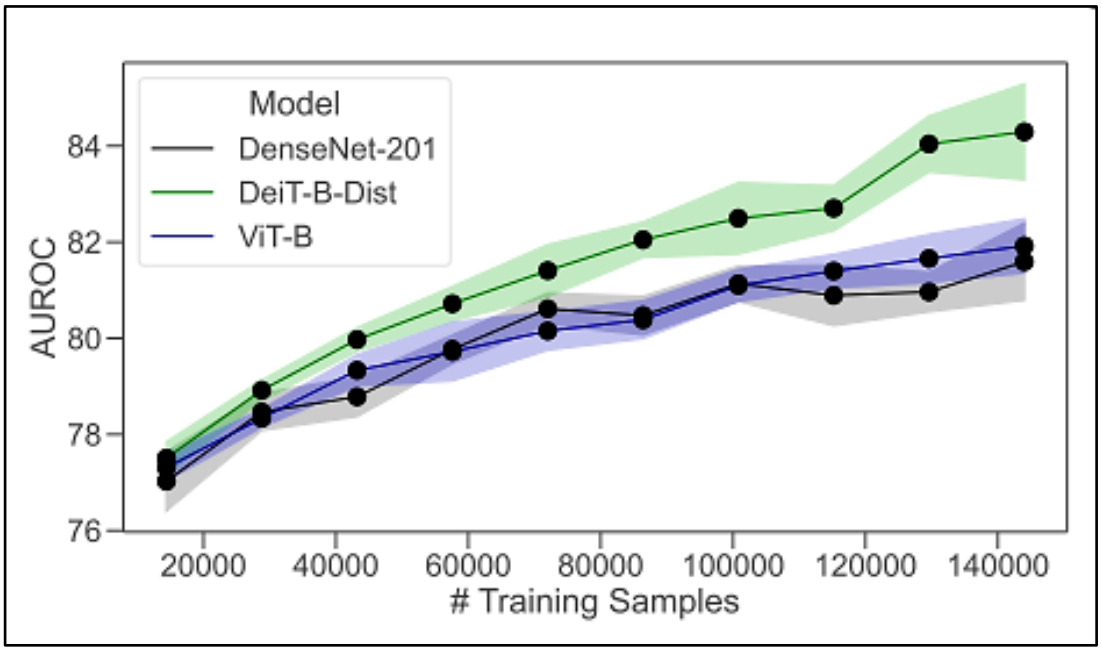

Рисунок 1 – График результатов обучения моделей
на наборе данных CheXpert

Automated Diagnosis of COVID-19 using X-ray Images and Deep Convolutional Neural Networks [9]

Исследователи использовали сверточную нейронную сеть для автоматического определения наличия COVID-19 на рентгеновских снимках грудной клетки. Авторы применяли набор данных из 798 изображений грудной клетки, из которых 413 были снимками пациентов с COVID-19, а 385 были отрицательными на COVID-19. Были использованы различные сетевые архитектуры, такие как ResNet, DenseNet и VGG, и выполнено сравнение их производительности при диагностике COVID-19. Исследователи получили высокую точность распознавания COVID-19 – 96,78%, когда ResNet50 был обучен на изображениях грудной клетки. Сравнение с другими

архитектурами показало, что архитектуры ResNet показали наиболее высокую производительность. На рисунке 2 представлены первые три изображения первой строки, которые являются образцами рентгеновских снимков с COVID-19, а остальные изображения представляют собой обычные рентгенограммы органов грудной клетки.

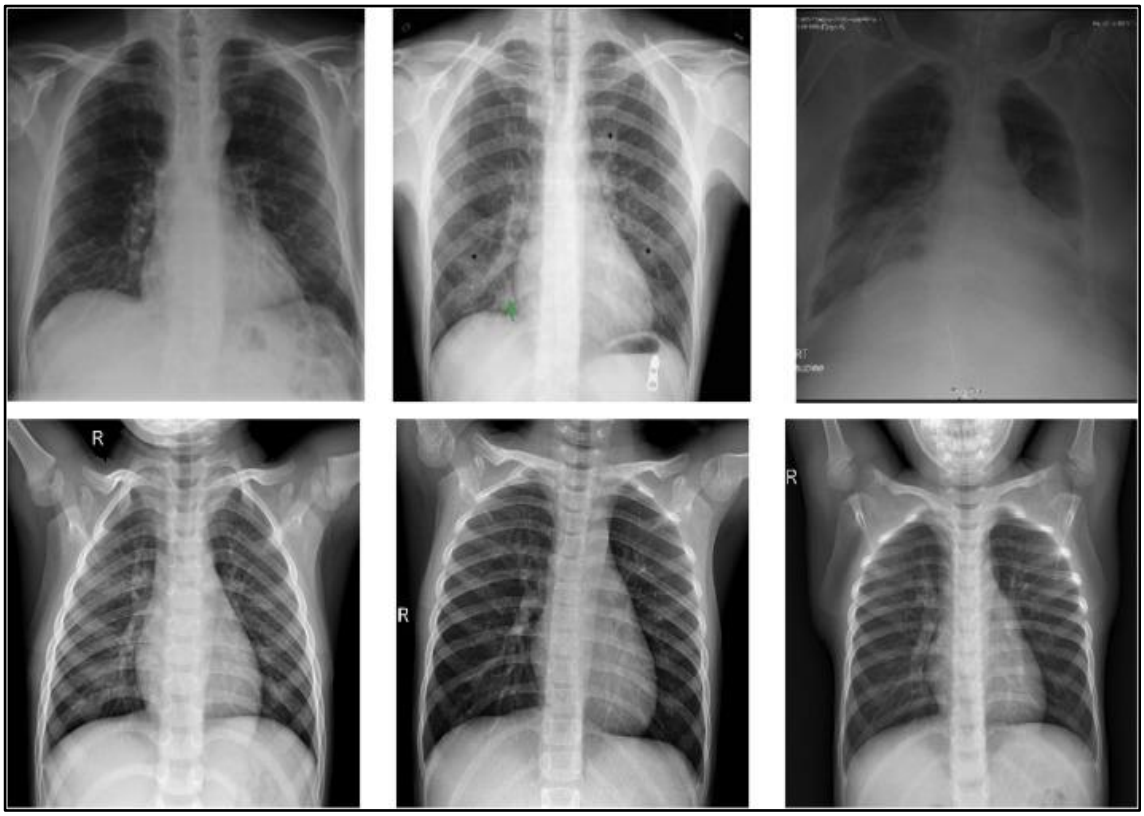

Рисунок 2 – Первые три изображения – рентгеновские снимки с COVID-19, остальные – снимки здоровых людей

Tuberculosis Diagnostics and Localization in Chest X-Rays via Deep Learning Models [4]

Авторы занимались обучением для автоматической диагностики туберкулеза на рентгеновских снимках грудной клетки. Использовались нейронные сети Faster R-CNN и ResNet-50, которые были обучены на большом наборе данных из 11 000 изображений грудной клетки, включая 4 208 изображений пациентов с туберкулом. Были проведены эксперименты для того, чтобы оценить производительность своих моделей. Результаты экспериментов показали высокую точность диагностики туберкулеза – 88,11%, а

также хорошую точность локализации туберкулезных поражений – 79,56%. В целом в данной статье показывается эффективность использования глубокого обучения для диагностики туберкулеза и локализации поражений на рентгеновских снимках грудной клетки. На рисунке 3 изображены результаты выявления заболеваний на тестовых изображениях для заболеваний, связанных с туберкулезом.

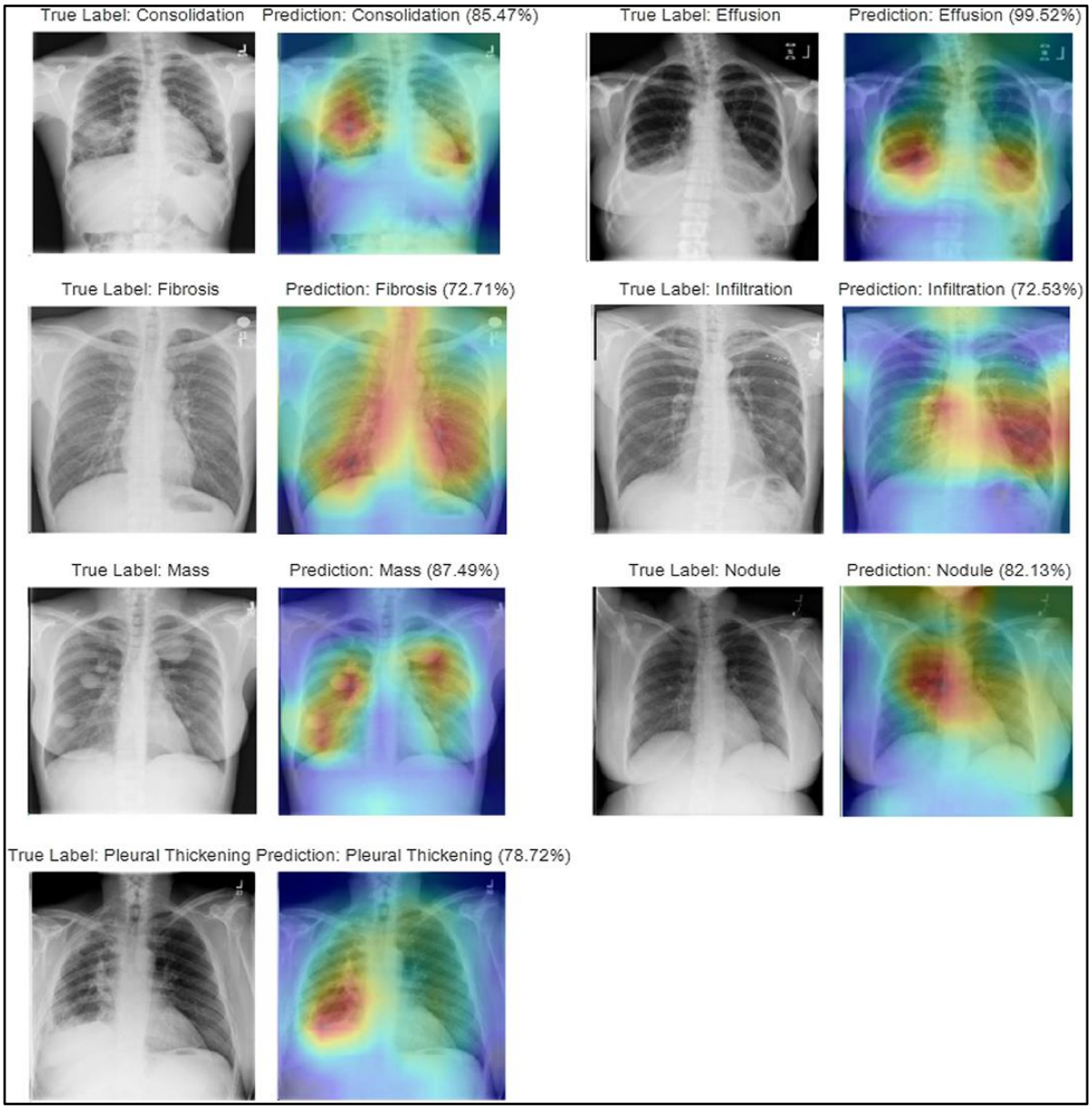

Рисунок 3 – Визуализация локализации заболевания на снимках

1.3. Анализ существующих решений для реализации проекта

ResNet (Residual Networks) [9] – это архитектура нейронной сети глубокого обучения, также известная как сеть остаточных блоков, она была

представлена компанией Microsoft в 2015 году. ResNet использует соединения быстрого доступа, которые пропускают один или несколько слоев и сопоставляют идентификаторы. ResNet достаточно легко оптимизировать и увеличивать точность при увеличении глубины, что с другими сетями добиться сложнее.

ViT (Vision Transformer) [7] – это модель машинного обучения, которая применяется для обработки изображений. Она представляет собой комбинацию трансформера, который обычно используется для обработки текста, и сверточных нейронных сетей, которые широко применяются для работы с изображениями. ViT преобразует изображение в последовательность векторов, которые затем подаются на вход трансформерной модели для обработки. Этот подход показал хорошие результаты в задачах компьютерного зрения и сильно изменил подход к обработке изображений в области машинного обучения.

DeiT (Data-efficient image Transformer) [1] – модель машинного обучения, которая представляет собой архитектуру трансформера, применяемую для обработки изображений. DeiT разработана с учетом эффективного использования данных и способна достигать высоких результатов на задачах компьютерного зрения даже при ограниченном количестве обучающих данных.

Выводы по первой главе

В первой главе была обозначена предметная область проекта, проведен анализ аналогичных проектов и рассмотрены решения, которые упростят реализацию проекта. При рассмотрении аналогичных проектов была подтверждена актуальность выбранной темы и разрабатываемого проекта.

2. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

2.1. Нейронная сеть

Нейронные сети – это математические модели, вдохновленные работой человеческого мозга и нервной системы. Они состоят из связанных между собой искусственных нейронов, которые обрабатывают входную информацию и передают ее дальше по сети. Разработка нейронных сетей началась еще в середине XX века, когда ученые пытались создать машины, обладающие искусственным интеллектом. В отличие от традиционных алгоритмических подходов, нейронные сети используют более сложные архитектуры, такие как сверточные или рекуррентные нейронные сети [5].

Сверточная нейронная сеть (Convolutional neural network) [6] – это нейронная сеть, состоящая из нескольких слоев, каждый из которых выполняет определенную функцию. Для извлечения признаков используется операция свертки, которая заключается в перемещении ядра свертки по пикселям изображения и вычислении скалярного произведения между ядром и пикселями, находящимися в его области. Операция пулинга (объединения) служит для уменьшения размера изображения путем удаления незначительной информации. После прохождения через сверточные слои, данные соединяются с полносвязными слоями, которые дают окончательные результаты прогнозирования или классификации.

Рекуррентные нейронные сети (Recurrent neural network) [10] – это тип нейронных сетей, предназначенных для обработки последовательных данных, таких как текст или речь. В отличие от традиционных нейронных сетей, RNN используют внутреннее состояние (память) для обработки текущих входных данных с учетом предыдущих. Основная идея RNN заключается в том, что они способны учитывать контекст и временные зависимости в данных. Это позволяет им эффективно моделировать и прогнозировать последовательности, что делает их незаменимыми в задачах обработки естественного языка, таких как машинный перевод, генерация текста и распознавание речи.

Архитектура трансформеров представляет собой развитие концепции нейронных сетей. Трансформеры [8] – это специальная архитектура глубоких нейронных сетей, представленная в 2017 году. Они были разработаны для эффективной обработки последовательных данных, таких как текст или видео, в отличие от традиционных рекуррентных нейронных сетей. Ключевой особенностью трансформера является использование механизма самовнимания (attention), который позволяет им моделировать долгосрочные зависимости в данных. Это делает трансформеры более эффективными и производительными [1] по сравнению с предыдущими архитектурами нейронных сетей.

2.2. Сегментация

Сегментация изображения по маске представляет собой процесс выделения конкретных областей или объектов на изображении путем применения масок, которые определяют форму и положение каждого объекта. Это важная технология в области компьютерного зрения, позволяющая автоматически выделять и анализировать определенные области изображения.

U-Net [8] – это сверточная нейронная сеть, которая была создана для сегментации биомедицинских изображений. Она подходит для задач, когда нужно сегментировать области изображения по классу, то есть создать маску, которая будет разделять изображение на несколько классов. Преимуществом архитектуры U-Net [8] является достижение высоких результатов при использовании небольшого количества данных.

Нейронная сеть включает в себя сверточную часть слева и развертночную часть справа, что создает архитектуру, напоминающую букву U. На каждом этапе количество каналов признаков увеличивается вдвое. Сверточная часть напоминает сверточную сеть, где содержатся два последовательных сверточных слоя размером 3×3, за которыми следует слой ReLU и пулинг с функцией максимума 2×2 с шагом 2. Каждый этап разверточной части включает в себя слой обратного пулинга, который расширяет карту

признаков, а затем следует свертка 2×2, уменьшающая количество каналов признаков. После этого идет конкатенация с соответственно обрезанной картой признаков из сокращенного пути, а затем две свертки 3×3, за каждой из которых следует ReLU. Обрезка необходима из-за потери граничных пикселей в каждой свертке. На последнем слое используется свертка 1×1 для преобразования каждого 64-компонентного вектора признаков до необходимого количества классов. Всего в сети содержится 23 сверточных слоя. На рисунке 4 изображена архитектура сети U-Net.

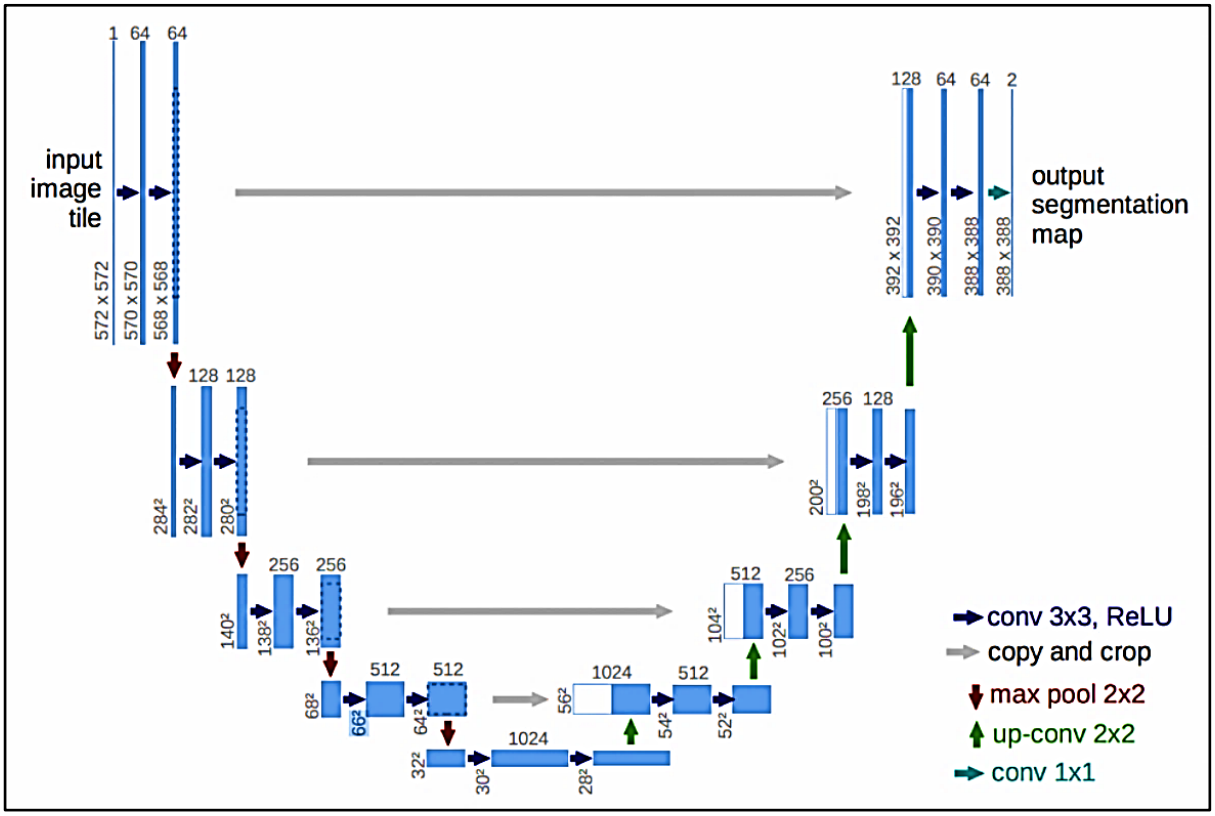

Рисунок 4 – U-Net архитектура

2.3. Архитектуры трансформеров в области компьютерного зрения

Трансформер [1] – это модель глубокого обучения, изначально разработанная для обработки последовательностей, таких как тексты и временные ряды. Основная идея трансформера заключается в том, что он способен анализировать взаимосвязи между элементами последовательности параллельно, в отличие от рекуррентных нейронных сетей, которые

обрабатывают последовательности пошагово. Трансформеры обычно состоят из кодировщика и декодировщика.

Для задачи классификации заболеваний на рентгеновских снимках были рассмотрены следующие модели трансформеров: ViT, Swin Transformer.

ViT (Vision Transformer) [1] – архитектура нейронной сети (рисунок 5), впервые примененная для обработки изображений с применением трансформеров, которые изначально были разработаны для обработки текста в естественном языке. ViT представляет собой концепцию, при которой изображение разбивается на патчи, которые затем подаются на вход в модель трансформера для обработки.

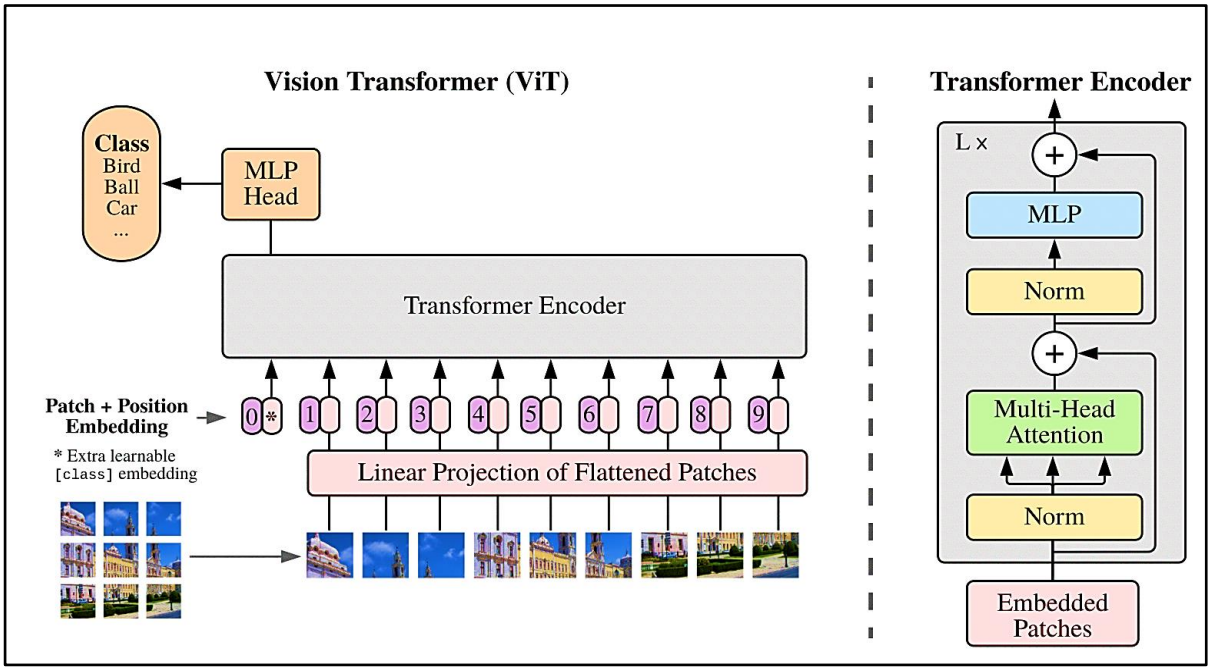

Рисунок 5 – Слева – принцип работы ViT, справа – блок трансформера ViT

Архитектура ViT основана на следующих шагах.

1. Изображение режется на фрагменты (patch).
2. Фрагменты (patch) подвергаются линейной проекции с помощью блока MLP (Multi layer perceptron) из одного или нескольких линейных слоев.

3. С полученными на выходе MLP векторами конкатенируются positional embeddings (кодирующие информацию о позиции path, как и в обычном трансформере для текста).

4. К полученным векторам добавляется еще один, который называют class embedding.

5. В финале этот специальный токен прогоняется через MLP и предсказывает классы.

На рисунке 6 представлена архитектура Swin Transformer, которая представляет собой модель, предназначенную для задач компьютерного зрения, таких как классификация изображений и обнаружение объектов.

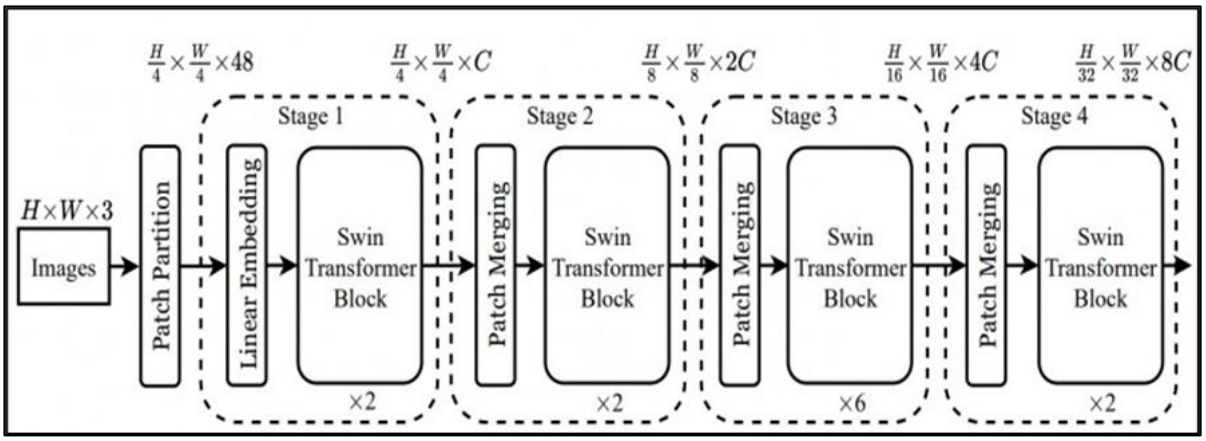

Рисунок 6 – Архитектура Swin Transformer

Модель состоит из четырех основных этапов:

1) входные изображения сначала проходят через раздел патчей и слой линейного внедрения, который разбивает изображение на непересекающиеся патчи и проецирует их в линейное пространство;

2) встроенные патчи проходят через первый блок Swin Transformer, который применяет операции self-attention (самовнимания) и прямой связи в локальной области окна размером 7×7;

3) проход через следующий блок Swin Transformer со смещенной схемой разделения окон для введения межоконных соединений;

4) выходные данные обрабатываются двумя блоками Swin Transformer с размерами окон 8×8 и 10×10 соответственно.

Сочетание иерархических признаковых карт и смещенных окон самовнимания делает Swin Transformer мощной и в то же время эффективной архитектурой для задач компьютерного зрения, таких как классификация изображений и обнаружение объектов.

2.3. Наборы данных

Набор данных VinBigData Chest X-ray для обучения модели классификации был взят с сайта Kaggle. Исходный набор данных рентгенографии грудной клетки состоит из 18 000 рентгеновских изображений в формате PNG с метками заболеваний легких. Содержит 15 классов. Изображения могут быть классифицированы как «No findings» («Нет результатов») или как один или несколько следующих классов заболеваний:

1) aortic enlargement (расширение аорты);
2) atelectasis (ателектаз);
3) calcification (кальцификация);
4) cardiomegaly (кардиомегалия);
5) consolidation (укрепление);
6) ILD (интерстициальное заболевание легких);
7) infiltration (инфильтрат);
8) lung opacity (помутнение легких);
9) nodule/mass (узелок/масса);
10) other lesion (другое поражение);
11) pleural effusion (плевральный выпот);
12) pleural thickening (плевральное утолщение);
13) pneumothorax (пневмоторакс);
14) pulmonary fibrosis (легочный фиброз).

На рисунке 7 представлен пример изображения из выбранного набора данных.

Рисунок 7 – Пример изображения из VinBigData Chest X-ray

На рисунке 8 представлена диаграмма распределения классов в исходном наборе данных.

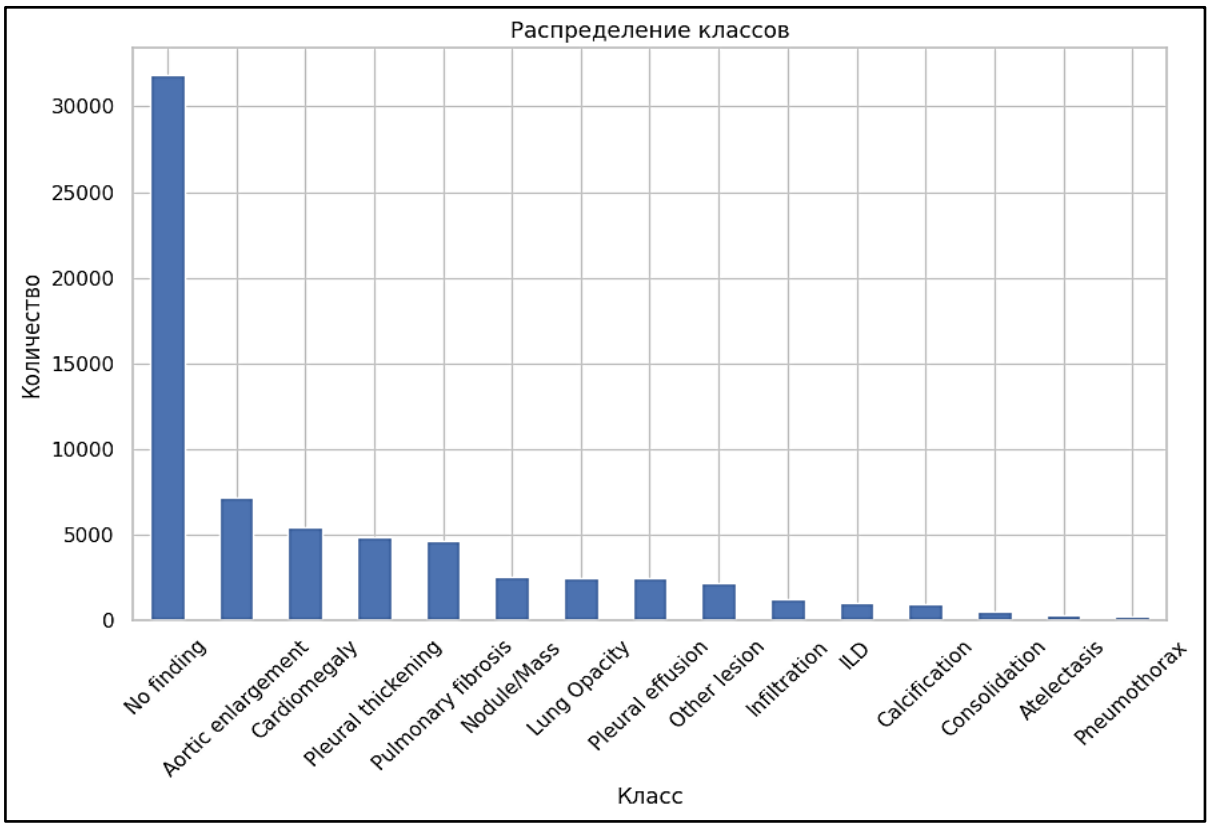

Рисунок 8 – Диаграмма распределение классов в исходном наборе данных

Далее набор данных был обработан следующим образом:

1) оставлено 8 000 снимков с классом «No findings»;

2) удалены снимки, относящиеся к классам заболеваний consolidation (укрепление), atelectasis (ателектаз), pneumothorax (пневмоторакс) так как их количество было меньше 1 000;

3) 8 454 снимка выбраны для обучения, 1 057 – для валидации, 1 057 – для тестирования.

На рисунке 9 представлена диаграмма распределения классов в обработанном наборе данных.

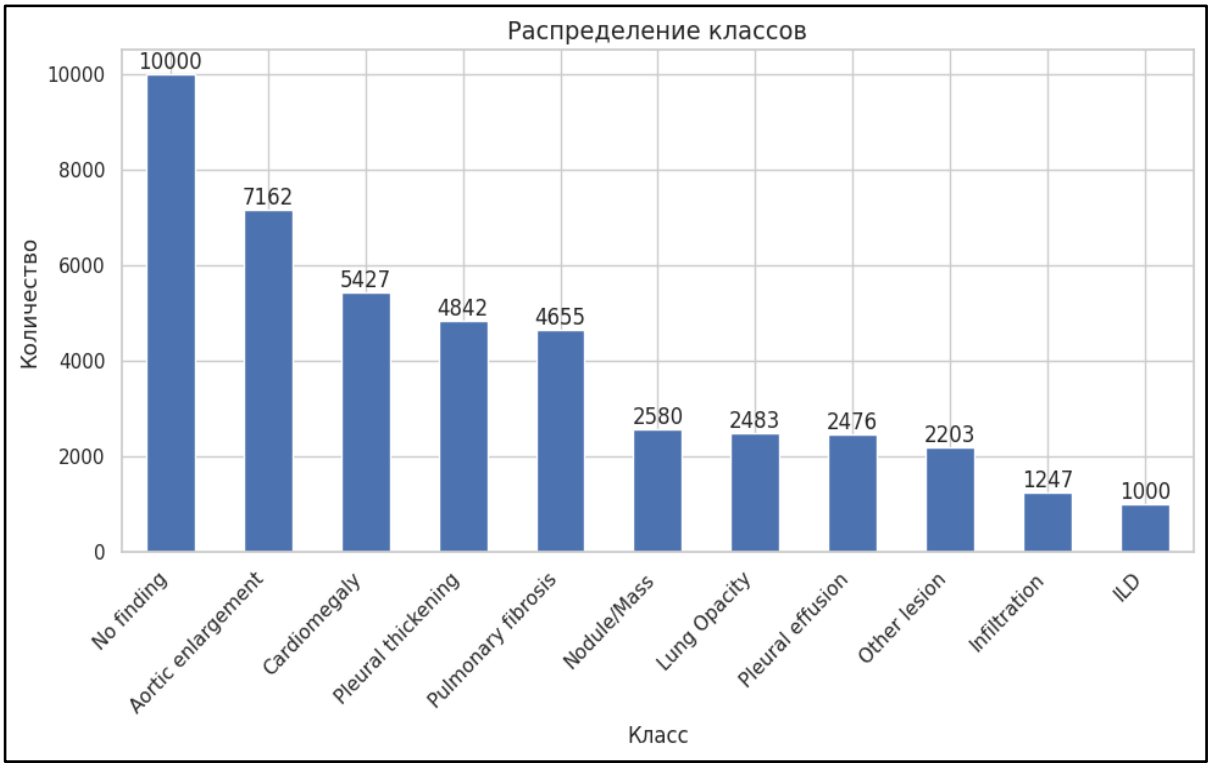

Рисунок 9 – Распределение классов в обработанном наборе данных

Набор данных Chest Xray Masks and Labels для сегментации изображений был взят с сайта Kaggle. Этот набор данных рентгенографии грудной клетки состоит из 896 рентгеновских изображений и 704 изображений масок сегментации в формате PNG.

На рисунках 10 и 11 представлены примеры изображений и масок из набора данных Chest Xray Masks and Labels соответственно.

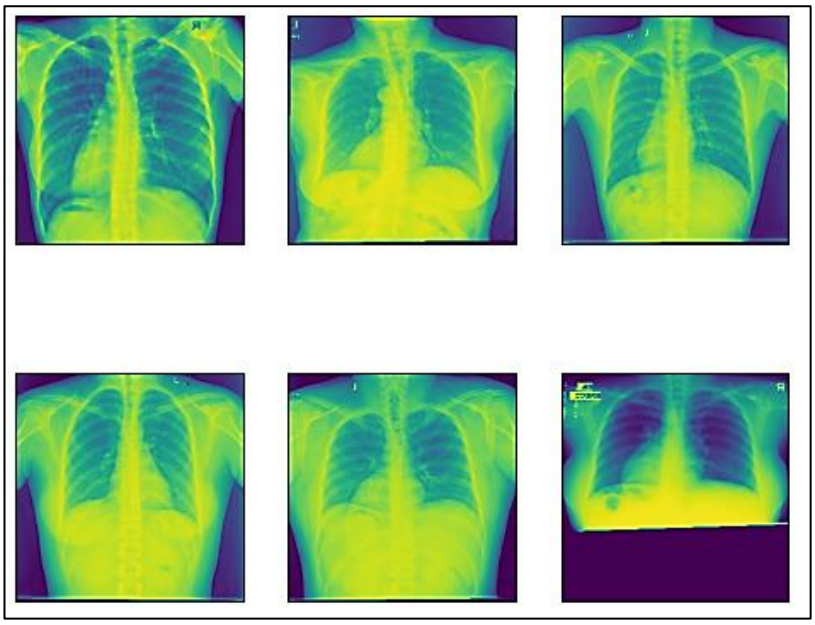

Рисунок 10 – Примеры изображений из Chest Xray Masks and Labels

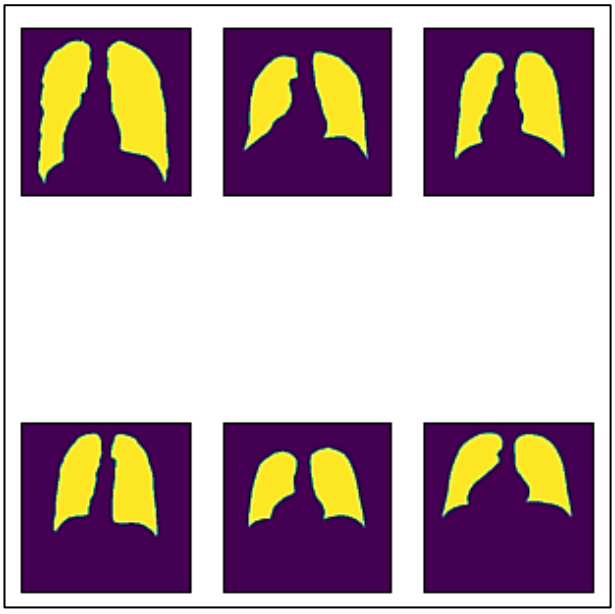

Рисунок 11 – Примеры масок из Chest Xray Masks and Labels

Выводы по второй главе

В этой главе были рассмотрены работы сверточных и рекуррентных нейронных сетей и архитектуры трансформеров для классификации медицинских снимков. Также был выбраны наборы данных для дальнейшего обучения и тестирования моделей для задач сегментации и классификации.

3. ПРОЕКТИРОВАНИЕ СИСТЕМЫ

3.1. Функциональные и нефункциональные требования

Функциональные требования

Функциональные требования [19] – условия, описывающие какое поведение должна предоставлять система.

Функциональные требования к разрабатываемой системе представлены ниже.

1. Система должна предоставлять возможность указывать путь до расположения изображения, которое находится в формате PNG.
2. Система должна изменять размер входящего изображения на тот, который использовался при обучении.
3. Система должна выводить на экран результат прогнозируемого заболевания.

Нефункциональные требования

Нефункциональные требования [19] представляют собой свойства или ограничения, накладываемые на систему.

Нефункциональные требования к разрабатываемой системе представлены ниже.

1. Система должна быть реализована на языке Python 3.10 [14].
2. Система должна использовать библиотеку PyTorch 2.3.0 [15] для работы с моделью нейронной сети.
3. Система должна обрабатывать одно изображение меньше чем за секунду.

3.2. Варианты использования системы

Для проектирования системы использовался UML (Unified Modeling Language) – унифицированный язык моделирования. В процессе проектирования была разработана диаграмма вариантов использования, которая описывает взаимодействие актера Пользователь с системой. На рисунке 12

изображена диаграмма вариантов использования системы выявления заболеваний на рентгеновских снимках.

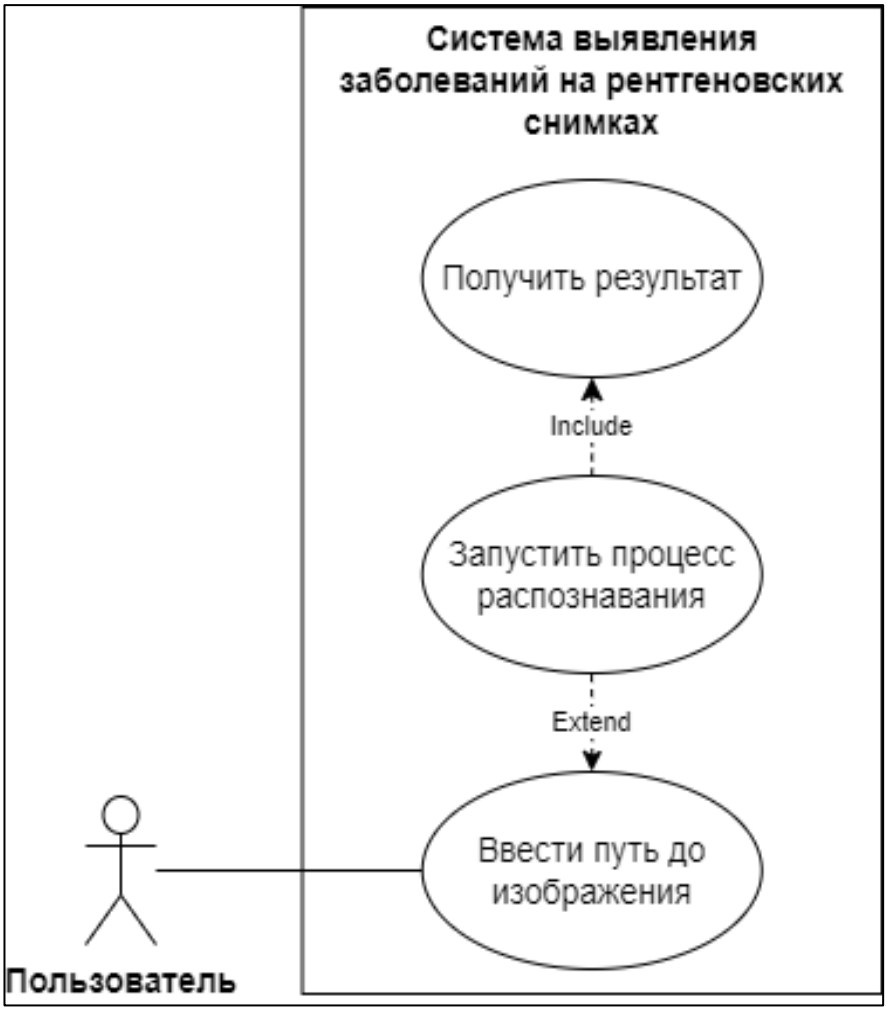

Рисунок 12 – Диаграмма вариантов использования

Основным актером, взаимодействующим с системой, является Пользователь.

Пользователь может совершать следующие действия.

1. Ввести путь до изображения. Пользователь вводит путь до изображения.

2. Запустить процесс распознавания. Система запускает работу обученной модели нейронной сети.

3. Получить результат. Система выводит результат распознавания.

Ниже в таблицах 1–3 представлены спецификации вариантов использования.

Таблица 1 – Спецификация варианта использования «Ввести путь до изображения»

Прецедент: Ввести путь до изображения
ID: 1
Краткое описание: Ввод пути до изображения рентгеновского снимка.
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: отсутствуют
Основной поток: 1. Пользователь вводит путь до изображения. 2. Пользователь нажимает «Enter».
Постусловия: Указан файл для распознавания.
Альтернативные потоки: отсутствуют

Таблица 2 – Спецификация варианта использования «Запустить процесс распознавания»

Прецедент: Запустить процесс распознавания
ID: 2
Краткое описание: Запуск процесса распознавания.
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: отсутствуют
Основной поток: Пользователь, указав путь до изображения и нажав «Enter», запускает процесс распознавания.
Постусловия: Запущен процесс распознавания.
Альтернативные потоки: отсутствуют

Таблица 3 – Спецификация варианта использования «Получить результат»

Прецедент: Получить результат
ID: 3
Краткое описание: Вывод изображения с выявленным классом заболевания.
Главные актеры: Пользователь
Второстепенные актеры: Нет

Окончание таблицы 3

Предусловия: Процесс распознавания завершен.
Основной поток: Пользователь получает результат классификации изображения.
Постусловия: На экран выводится изображение с выявленным классом заболевания.
Альтернативные потоки: отсутствуют

Диаграмма деятельности

На рисунке 13 изображена диаграмма деятельности системы выявления аномалий на рентгеновских снимках. Пользователь указывает путь до изображения, после чего система считывает путь до изображения и производит предварительную обработку входного изображения, в том числе изменяет размер изображения на тот, который использовался при обучении модели. Далее запускается работа модели нейронной сети, результат работы которой выводится в виде изображения с прогнозируемыми классами заболеваний. Система выводит результат классификации. Пользователь в свою очередь просматривает результат.

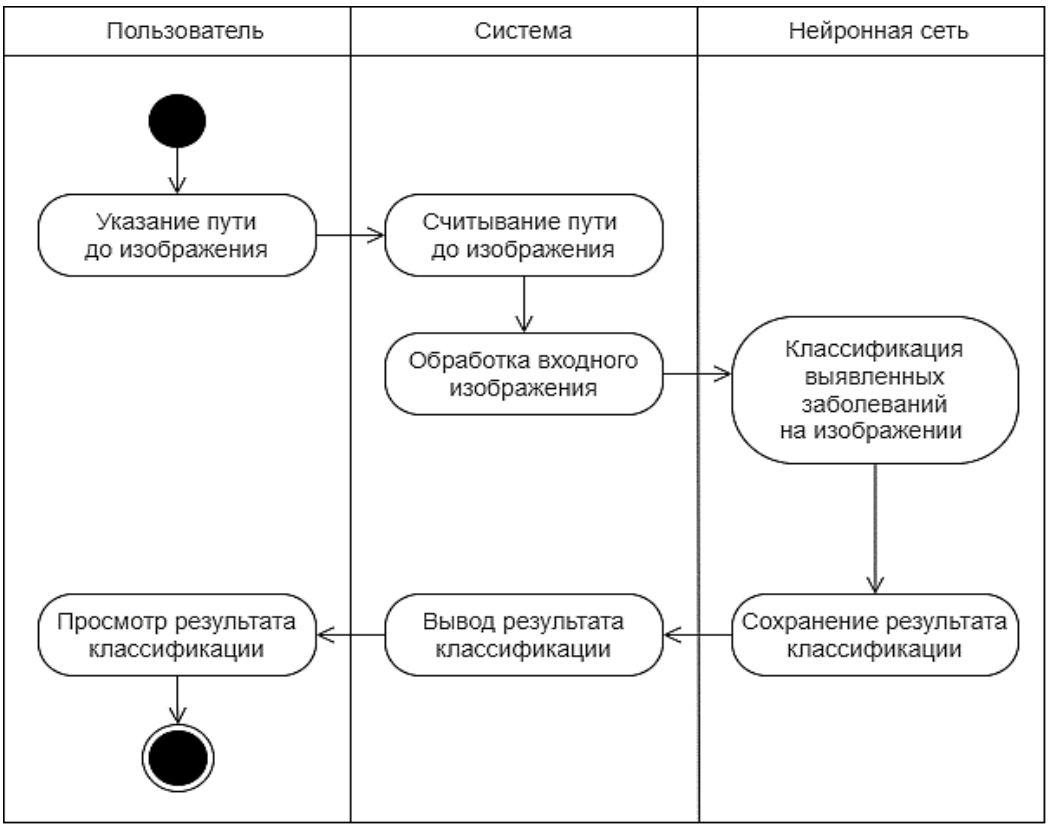

Рисунок 13 – Диаграмма деятельности

3.3. Топология нейронной сети

Для классификации заболеваний на изображениях была реализована архитектура Swin Transformer предобученной модели из библиотеки PyTorch. Топология этой нейронносой сети состоит из четырех стадий и заключительного слоя.

1. Первая стадия разбивает входное изображение на непересекающиеся патчи размером 4×4, далее патчи линейно проецируются в пространство размерности 48, проходят через один блок Swin Transformer с размером окна 7×7, и размер признаковых карт уменьшается в 2 раза (до 48×48).

2. Вторая стадия проводит признаковые карты через один блок Swin Transformer с размером окна 7×7 и схемой смещения окон, затем размер признаковых карт уменьшается в 2 раза (до 24×24) и количество каналов увеличивается до 96.

3. В третьей стадии после прохода через блок трансформера размер признаковых карт уменьшается в 2 раза (до 12×12) и количество каналов увеличивается до 192.

4. В четвертой стадии признаковые карты проходят через два блока трансформера с размерами окон 8×8 и 10×10 соответственно, размер признаковых карт уменьшается в 6 раз (до 6×6), количество каналов увеличивается до 384.

5. Заключительный слой содержит глобальный пулинг для получения вектора признаков фиксированного размера и полносвязный слой для классификации изображений.

Выводы по третьей главе

В третьей главе были определены функциональные и нефункциональные требования для системы, разработаны диаграмма вариантов использования и диаграмма деятельности, произведено проектирование нейронной сети для классификации заболеваний на рентгеновских снимках, обучение которой представлено в следующем разделе.

4. ПРАКТИЧЕСКАЯ ЧАСТЬ

4.1. Программные средства реализации

Обучение нейронной сети проводилось на облачной платформе Google Colaboratory [3], которая предоставляет вычислительные ресурсы для обучения глубоких нейронных сетей, доступ к GPU и TPU. Оперативная память системы составляет 25,5 ГБ, оперативная память графического процессора – 15 ГБ, диск – 166,8 ГБ, ускоритель Nvidia T4 GPU. Для реализации программной части системы использовался язык программирования Python 3.10 [15]. Были использованы следующие библиотеки Python.

Numpy [13]

Это библиотека языка программирования Python, которая предоставляет мощные возможности для работы с многомерными массивами и матрицами.

Pandas [14]

Библиотека для анализа и обработки данных. Она предоставляет широкие возможности для чтения, записи, фильтрации, преобразования и агрегации данных, а также для работы с временными рядами, таблицами и другими структурами данных.

Itertools [11]

Встроенная библиотека языка Python, которая предоставляет функции для выполнения операций с итераторами.

Sklearn [17]

Библиотека машинного обучения на языке программирования Python, которая предоставляет широкий спектр инструментов для решения задач обучения с учителем и без учителя, выборов признаков и визуализации данных.

Matplotlib [12]

Используется для создания разнообразных графиков, диаграмм, распределений, спектрограмм, и других видов визуализации данных. Библиотека Matplotlib предоставляет гибкие инструменты для создания

высококачественных графиков, включая возможность настройки цветов, стилей линий и маркеров, создание легенд, добавление текста и аннотаций, и многое другое.

PyTorch [16]

Библиотека, которая облегчает работу с глубокими нейронными сетями, обеспечивает простой и интуитивно понятный интерфейс для создания, обучения и оценки моделей обучения.

4.2. Обучение нейронной сети

Обучение проходило в облачном сервисе Google Colaboratory Pro. В Swin Transformer использованы: оптимизатор – Adam, функция потерь – `BCEWithLogitsLoss` с параметром `pos_weight`. Нейронная сеть выдает прогнозируемые значения (логиты) для каждого класса. Функция `BCEWithLogitsLoss` сравнивает прогнозируемые логиты нейронной сети с истинными метками. Она вычисляет, насколько сильно прогноз отличается от истины. Затем она использует эту разницу, чтобы вычислить значение функции потерь. Цель обучения – минимизировать эту функцию потерь, чтобы заставить нейронную сеть делать более точные прогнозы. Параметр `pos_weight`, который позволяет придавать больший вес ошибкам в редких классах. На рисунке 14 представлены веса для каждого класса заболевания.

```
{'No finding': 0.478125, 'Cardiomegaly': 0.7048092868988391,
'Aortic enlargement': 0.5340686958950014, 'Pleural thickening': 0.7899628252788105,
'ILD': 3.825, 'Nodule/Mass': 1.4825581395348837,
'Pulmonary fibrosis': 0.8216970998925887, 'Lung Opacity': 1.5404752315747081,
'Other lesion': 1.7362687244666364, 'Infiltration': 3.067361668003208,
'Pleural effusion': 1.5448303715670437}
```

Рисунок 14 – Веса классов

Для обучения модели выявления аномалий на рентгеновских снимках было использовано 8 454 изображений, которые содержат метки 11 классов заболеваний. Размер обучающей выборки в процентном соотношении – 80%,

от основного набора данных. Размер каждого изображения 224 на 224. По результатам экспериментов обучения была выбрана модель, обученная на 30 эпохах со следующими метриками: accuracy – 0,923, precision – 0,715, recall – 0,611. На рисунках 15–17 представлены графики метрик модели.

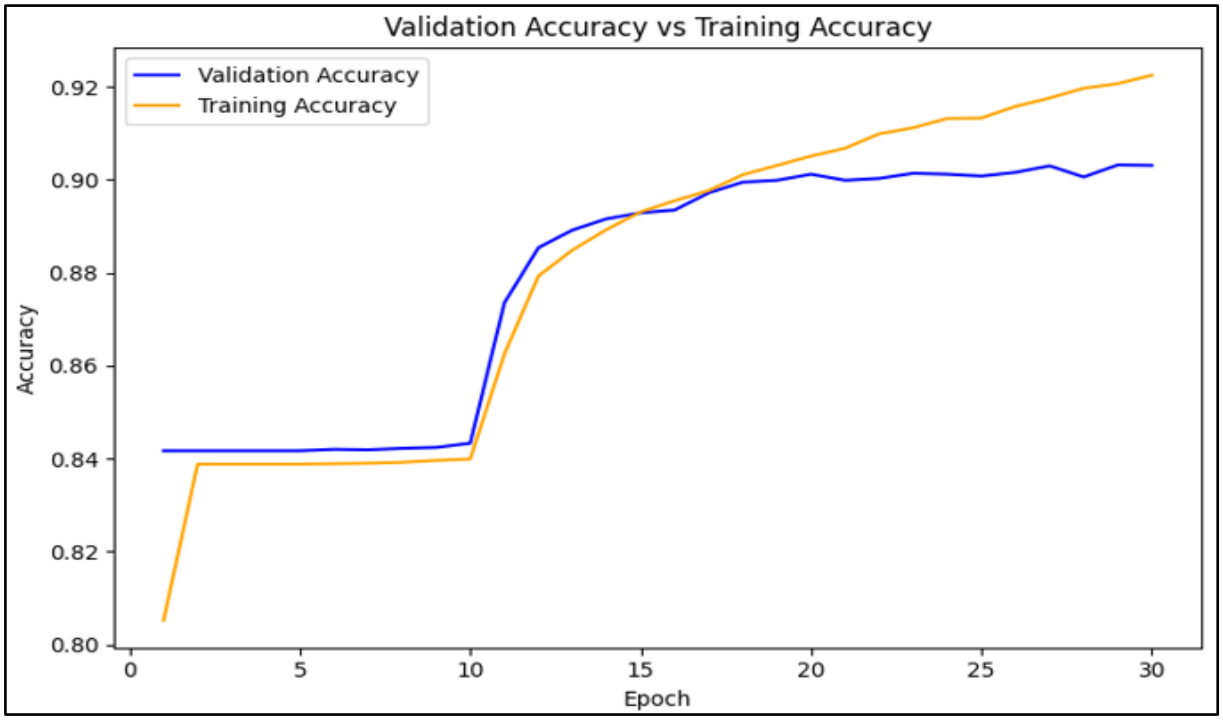

Рисунок 15 – График метрики accuracy

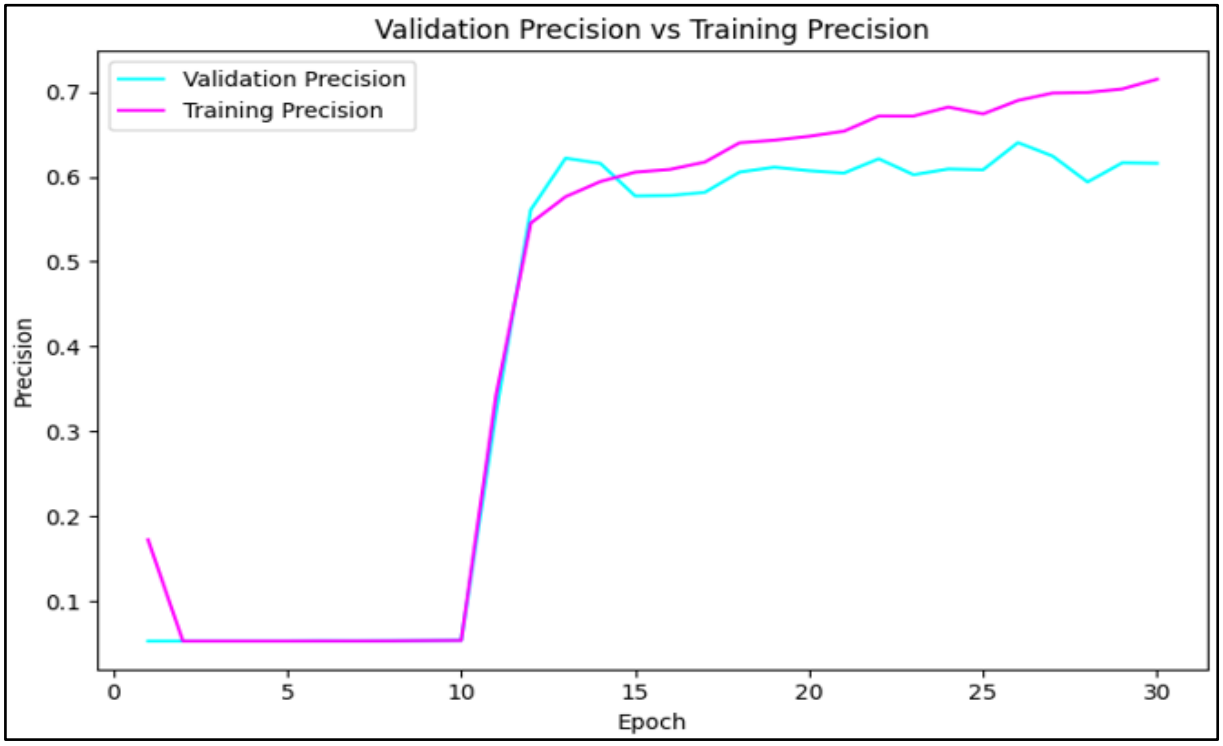

Рисунок 16 – График метрики precision

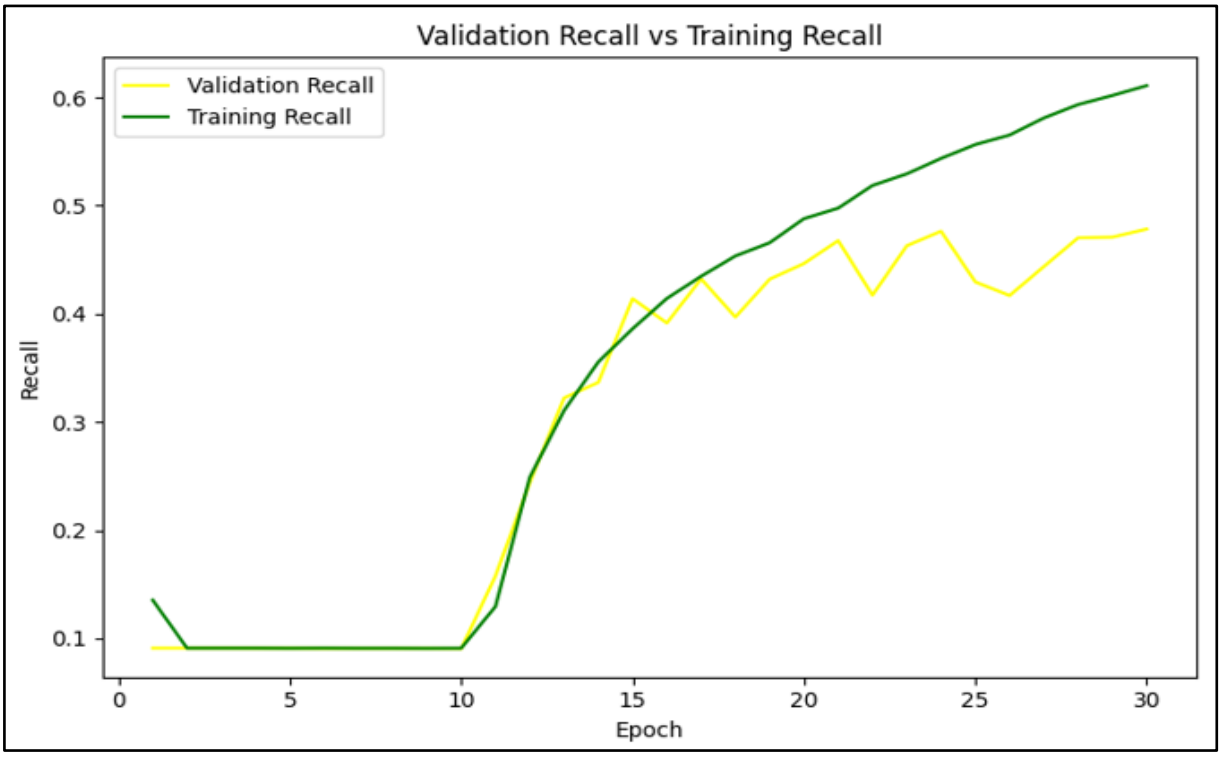

Рисунок 17 – График метрики recall

Обучение модели для сегментации легких на рентгеновских снимках проводилось в два этапа в течение 15 и 30 эпох. В первый этап количество эпох составило 15, размер батча – 20 и обучение заняло 30 минут. Второй этап обучения длился в течение 30 эпох, размер батча – 16, и занял 58 минут. Также применялся обратный вызов ReduceLROnPlateau из библиотеки Keras, который уменьшает скорость обучения (learning rate) в процессе обучения нейронной сети в случае, если метрика, указанная в параметре monitor (в данном случае, loss), перестает улучшаться после определенного количества эпох. Модель компилируется с помощью метода compile, где используется оптимизатор – adam, функция потерь – binary crossentropy и метрика качества модели – accuracy. Результаты обучения представлены на рисунке 18.

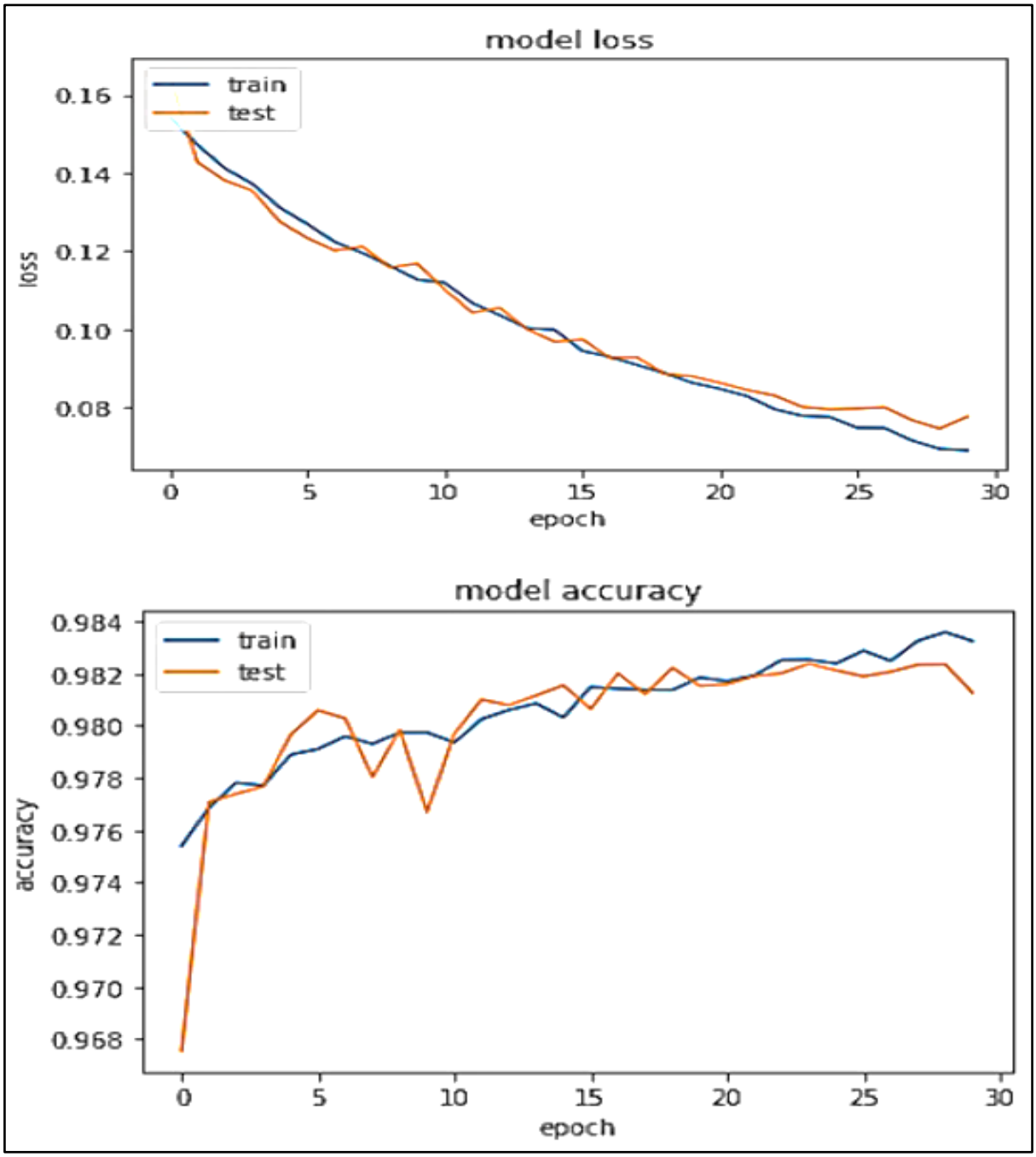

Рисунок 18 – График функции потерь (сверху), точность модели (снизу)

4.3. Вид приложения

Для взаимодействия пользователя с системой было использовано консольное приложение, которое позволяет пользователю указать путь до изображения. После завершения работы приложение выводит на экран результаты классификации заболеваний на введенном изображении. Вид и результат работы приложения представлен на рисунке 19. Программная реализация консольного приложения представлена в приложении.

```
Enter the path to the input image: /content/1b3e916aa92b328e17f24d1c892cb749.png
Predicted Diseases:
Aortic enlargement: 0.97
Pulmonary fibrosis: 0.02
Pleural thickening: 0.01
Nodule/Mass: 0.00
Calcification: 0.00
Cardiomegaly: 0.00
Other lesion: 0.00
Lung Opacity: 0.00
Infiltration: 0.00
ILD: 0.00
Pleural effusion: 0.00
Atelectasis: 0.00
Consolidation: 0.00
Pneumothorax: 0.00
```

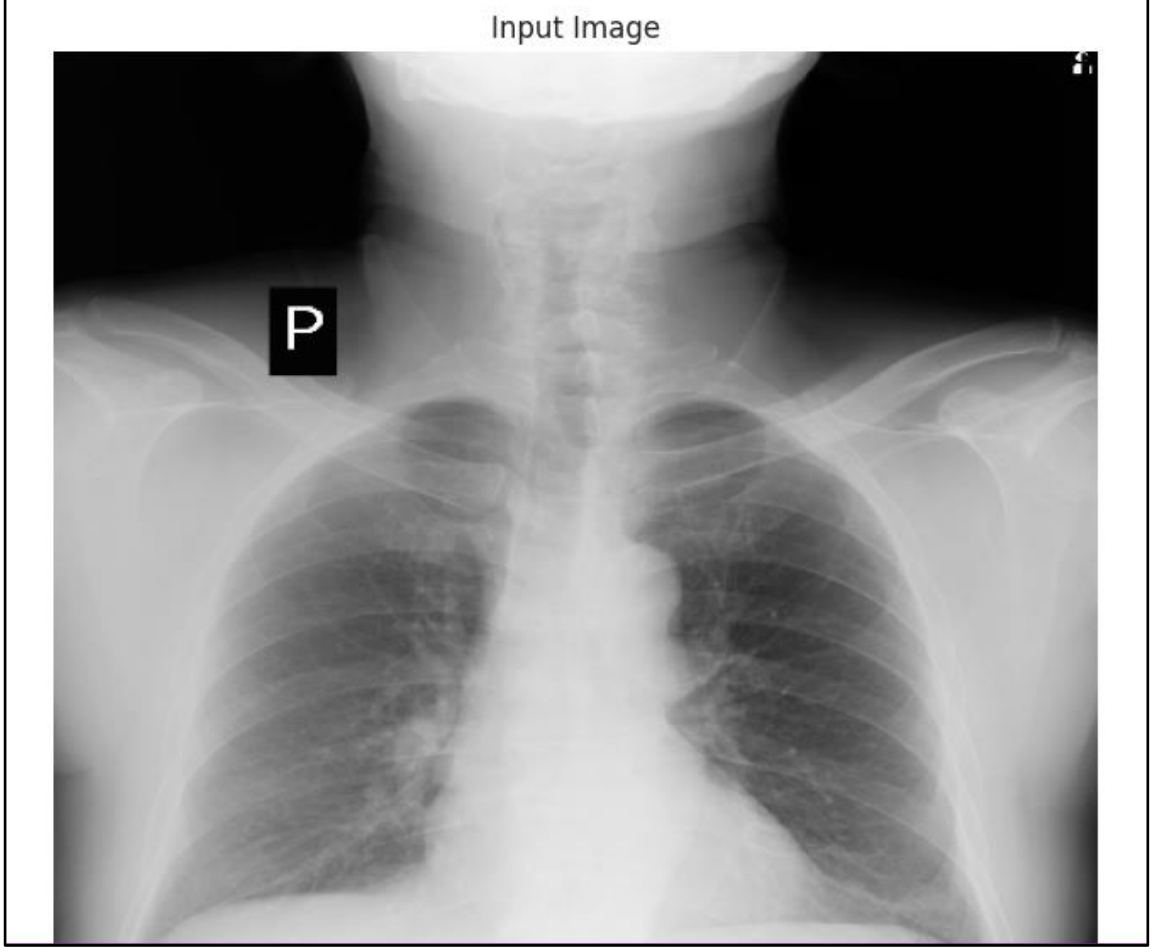

Рисунок 19 – Вид и результат работы приложения

Выводы по четвертой главе

Были приведены программные средства реализации, обучены модели нейронных сетей, результаты обучения были изображены на графиках. Разработано приложение, предоставлен вид и результат его работы.

5. ТЕСТИРОВАНИЕ

5.1. Тестирование нейронной сети

Тестирование модели на основе трансформера проводилось на изображениях, не содержащихся в обучающей и валидационной выборах. Модель продемонстрировала точность на уровне 90,11% на тестовой выборке. По результатам тестирования была выведена таблица метрик качества для классификации каждого класса заболевания. В таблице 4 представлен результат тестирования работы модели нейронной сети.

Таблица 4 – Метрики качества для классификации каждого класса заболевания

Class name	precision	recall	f1-score
Aortic enlargement	0,86	0,67	0,75
Cardiomegaly	0,81	0,69	0,75
ILD	0,51	0,50	0,51
Infiltration	0,47	0,54	0,50
Lung Opacity	0,59	0,42	0,49
No finding	0,90	0,95	0,92
Nodule/Mass	0,34	0,12	0,18
Other lesion	0,31	0,13	0,18
Pleural effusion	0,77	0,26	0,39
Pleural thickening	0,67	0,35	0,46
Pulmonary fibrosis	0,78	0,19	0,30

Тестирование модели для сегментации проводилось на изображениях, не содержащихся в обучающей выборке. Модель продемонстрировала точность на уровне 98,08% на тестовой выборке. По результатам тестирования были выведены сгенерированные маски по форме легких по рентгеновским снимкам.

На рисунке 20 представлены примеры сгенерированных масок области легких по рентгеновскому снимку из тестовой выборки набора данных.

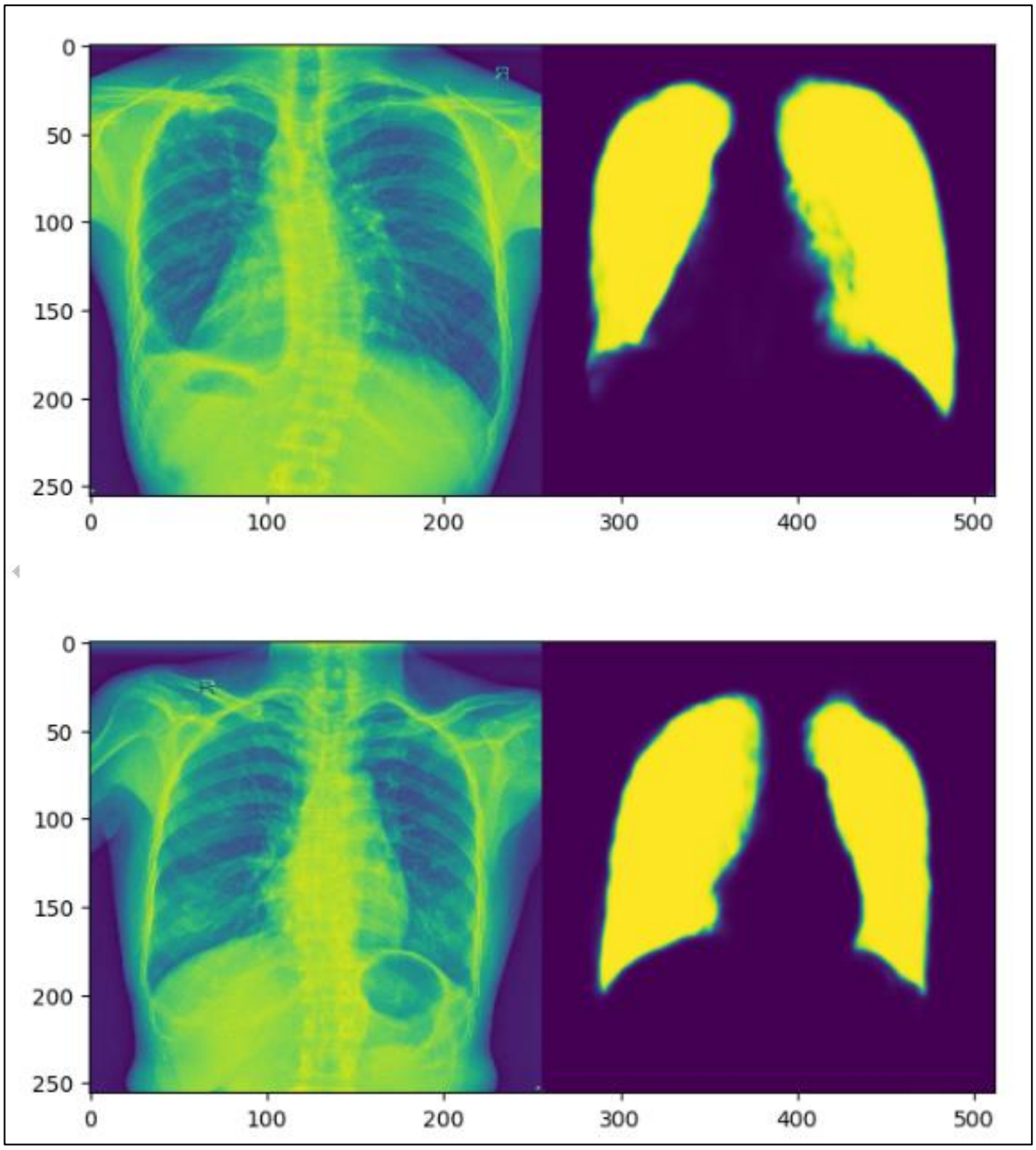

Рисунок 20 – Примеры сгенерированных масок

Эксперимент над обучением нейронной сети

Для того, чтобы получить значение оптимального количества эпох для обучения нейронных сетей, был проведен эксперимент. Обучение каждой нейронной сети проводилось несколько раз с одинаковым значением параметров, кроме количества эпох, оно изменялось с 10 до 40. Результаты экспериментов представлены в таблице 5.

Таблица 5 – Результаты экспериментов обучения нейронной сети

Кол-во эпох	Время обучения/ мин.	Accuracy	Precision	Recall	F1
15	60	0,887	0,572	0,322	0,385
30	120	0,923	0,715	0,611	0,642
40	250	0,911	0,706	0,356	0,413

5.2. Функциональное тестирование приложения

Функциональное тестирование [20] – это процесс тестирования программного обеспечения, направленный на проверку соответствия системы или ее компонентов заданным функциональным требованиям. Результаты тестирования приложения представлены в таблице 6.

Таблица 6 – Результаты функционального тестирования приложения

№	Предусловие	Действие	Ожидаемый результат	Результат
1	Приложение запущено, указан путь до изображения	Приложение должно обработать изображение, изменив его размер	Приложение обработало входное изображение	Пройден
2	Приложение запущено, указан путь до изображения, запущена классификация заболеваний на изображении	Приложение должно запросить путь до изображения для дальнейшей работы	Приложение сохранило результат классификации	Пройден
3	Приложение закончило процесс классификации заболеваний на изображении	После завершения классификации приложение должно вывести результат на экран	Приложение вывело результат классификации на экран	Пройден

Выводы по пятой главе

В данной главе было проведено тестирование разработанной нейронной сети. На основе подготовленных функциональных тестов было проведено тестирование разрабатываемого приложения. Все тесты были успешно пройдены.

ЗАКЛЮЧЕНИЕ

В соответствии с целью данной работы был проведен обзор научной литературы и аналогов. Были приведены теоретические сведения об архитектурах на основе трансформера. Были сформулированы функциональные и нефункциональные требования к разрабатываемому приложению. Также была разработана диаграмма вариантов использования и диаграмма деятельности. В соответствии с целью данной работы была реализована топология нейронной сети для классификации заболеваний на рентгеновском снимке, проведено обучение, тестирование. Разработано приложение. Также на основе подготовленных функциональных тестов было проведено тестирование разрабатываемого приложения. Приложение успешно прошло все тесты.

В рамках данной работы была разработана система для выявления аномалий на рентгеновских снимках. При этом были решены следующие задачи.

1. Произведен анализ предметной области.
2. Реализована выбранная топология искусственной нейронной сети, обучена ее и протестирована.
3. Разработать систему выявления аномалий на рентгеновских снимках.
4. Проведено тестирование системы.

В будущем планируется провести работу по повышению точности выявления аномалий на рентгеновских снимках легких, разработке мобильного приложения для удобства использования, а также поиску и сравнению возможных других алгоритмов классификации и распознавания медицинских снимков для наилучшей работы модели. Ожидается, что дальнейшее развитие системы способствует улучшению качества медицинской помощи и обеспечит более удобный доступ к современным технологиям диагностики.

ЛИТЕРАТУРА

1. Behrendt F. Data-Efficient Vision Transformers for Multi-Label Disease Classification on Chest Radiographs / F. Behrendt, D. Bhattacharya, J. Krüger, R. Opfer, A. Schlaefer. // Cornel University, 2022. – 4 p.

2. Eshraghi M. A. COV-MobNets: a mobile networks ensemble model for diagnosis of COVID-19 based on chest X-ray images / M. A. Eshraghi, A. Ayatollahi, B. Shokouhi Shahriar. // ResearchGate, 2023. – 11 p.

3. Google Colaboratory [Электронный ресурс] URL: https://colab.research.google.com/?hl=ru_RU (дата обращения 10.05.2024 г.).

4. Guo R. Tuberculosis Diagnostics and Localization in Chest X-Rays via Deep Learning Models / R. Guo, K. Passi, K. Jain Chakresh. // Frontiers, 2020. – 17 p.

5. He K. Deep Residual Learning for Image Recognition / K. He, X. Zhang, S. Ren, J. Sun. // Cornel University, 2015. – 12 p.

6. Huang G. Densely Connected Convolutional Networks / G. Huang, Z. Liu, Laurens van der Maaten, K.Q. Weinberger. // Cornel University, 2018. – 9 p.

7. Ivanova O.N. Intermediate Fusion Approach for Pneumonia Classification on Imbalanced Multimodal Data. / O.N. Ivanova, A.V. Melekhin, E.V. Ivanova, S. Kumar, M.L. Zymbler. // Bulletin of the South Ural State University, 2023. – PP. 19–30 pp.

8. Liu W. Automatic lung segmentation in chest X-ray images using improved U-Net/ W. Liu, J. Luo, Y. Yang, W. Wang, J. Deng, L. Yu. // Scientific Reports, 2021. – 11 p.

9. Saha P. EMCNet: Automated COVID-19 diagnosis from X-ray images using convolutional neural network and ensemble of machine learning classifiers / P. Saha, M.S. Sadi, M.M. Islam. // ScienceDirect, 2020. – 10 p.

10. Воронина В.В. Теория и практика машинного обучения: учебное пособие. / В.В. Воронина, А.В. Михеев, Н.Г. Ярушкина, К.В. Святов // Ульяновск: УлГТУ, 2017. – 290 с.

11. Документация Itertools. [Электронный ресурс] URL: https://docs.python.org/3/library/itertools.html (дата обращения:13.05.2024 г.).

12. Документация Matplotlib 3.7.1. [Электронный ресурс] URL: https://matplotlib.org/stable/index.html (дата обращения: 13.05.2024 г.).

13. Документация NumPy. [Электронный ресурс] URL: https://numpy.org/doc/stable/ (дата обращения: 13.05.2024 г.).

14. Документация Pandas. [Электронный ресурс] URL: https://pandas.pydata.org/docs/ (дата обращения: 13.05.2024 г.).

15. Документация Python. [Электронный ресурс] URL: https://www.python.org/ (дата обращения: 10.05.2024 г.).

16. Документация PyTorch 2.3.0. [Электронный ресурс] URL: https://pytorch.org/docs/2.3/ (дата обращения: 13.05.2024 г.).

17. Документация Scikit-learn. [Электронный ресурс] URL: https://scikit-learn.org/0.21/documentation.html (дата обращения: 13.05.2024 г.).

18. Матиас Х. Рентгенологическое исследование грудной клетки. Практическое руководство. Атлас. // М.: Медицинская литература, 2023. – 224 с.

19. Новиков Ф. А. Учебно-методическое пособие по дисциплине «Анализ и проектирование на UML». // Спб: Санкт-Петербурский университет информационных технологий, механики и оптики, 2007. – 286 с.

20. Старолетов С. М. Основы тестирования и верификации программного обеспечения: учебное пособие. // Санкт-Петербург: Лань, 2020. – 344 с.

ПРИЛОЖЕНИЕ. Листинги кода

Листинг 1 – Код программного модуля «detection_of_anomalies_on_x-rays.py»

```
from google.colab import drive
drive.mount('/content/drive')

import os
import zipfile
import numpy as np
import pandas as pd

zip_file = '/content/drive/MyDrive/archive (1).zip'

z = zipfile.ZipFile(zip_file, 'r')
z.extractall()

train_df = pd.read_csv('/content/vinbigdata/train.csv')

train_df.shape

train_df.head()

test_df = pd.read_csv('/content/vinbigdata/test.csv')

test_df.shape

test_df.head()

#train_df = train_df[train_df["class_name"] != "No finding"]

train_df.shape

train_df.head()

unique_image_ids = train_df['image_id'].nunique()

print(f"Количество уникальных значений в колонке 'image_id':
{unique_image_ids}")

import pandas as pd
import matplotlib.pyplot as plt

# Подсчет количества элементов для каждого класса
class_counts = train_df['class_name'].value_counts()

# Создание гистограммы
class_counts.plot(kind='bar', figsize=(10, 6))
plt.xlabel('Класс')
plt.ylabel('Количество')
plt.title('Распределение классов')
plt.xticks(rotation=45)
plt.show()

# Оставляем все строки, кроме первых 10000 строк
no_finding_rows = train_df[train_df['class_name'] == "No finding"]
no_finding_rows = no_finding_rows.iloc[8000:]
new_train_df = train_df[~train_df.index.isin(no_finding_rows.index)]
print(f"Количество строк после удаления: {len(new_train_df)}")

from itertools import chain
labels = np.unique(list(chain(*train_df['class_name'].map(lambda x:
x.split('|')).tolist())))
labels = [x for x in labels if len(x) > 0]
```

```
labels

class_counts = new_train_df['class_name'].value_counts()

# Создание списка классов с количеством больше или равно 1000
valid_classes = class_counts[class_counts >= 1000].index.tolist()

# Удаление строк с классами, количество которых меньше 1000
filtered_data_train =
new_train_df[new_train_df['class_name'].isin(valid_classes)]
print(f"Количество строк после удаления классов:
{len(filtered_data_train)}")

# Подсчет количества элементов для каждого класса
class_counts = filtered_data_train['class_name'].value_counts()

# Создание гистограммы
fig, ax = plt.subplots(figsize=(10, 6))
class_counts.plot(kind='bar', ax=ax)
ax.set_xlabel('Класс', fontsize=12)
ax.set_ylabel('Количество', fontsize=12)
ax.set_title('Распределение классов', fontsize=14)
ax.set_xticklabels(class_counts.index, rotation=45, ha='right')

# Подписывание количества на каждой колонке
for i, v in enumerate(class_counts.values):
    ax.text(i, v, str(v), ha='center', va='bottom')

plt.tight_layout()
plt.show()

labels = np.unique(list(chain(*filtered_data_train['class_name'].map(lambda
x: x.split('|')).tolist())))
labels = [x for x in labels if len(x) > 0]
labels

# размеры найденных заболеваний
import seaborn as sns
sns.scatterplot(data=filtered_data_train, x="width",
y="height",hue="class_name", palette = 'viridis')
plt.legend(bbox_to_anchor=( 1.02 , 1 ), loc='upper left', borderaxespad= 0
)

unique_image_ids = filtered_data_train['class_id'].unique()
print(unique_image_ids)

pip install --upgrade albumentations

pip install nvidia-smi

!pip install monai thop pydicom

# Commented out IPython magic to ensure Python compatibility.
!python -c "import monai" || pip install -q "monai-weekly[pillow, tqdm]"
!python -c "import matplotlib" || pip install -q matplotlib
# %matplotlib inline

import os
import shutil
import tempfile
import matplotlib.pyplot as plt
import PIL
```

```
import torch
import numpy as np
from sklearn.metrics import classification_report

from monai.apps import download_and_extract
from monai.config import print_config
from monai.data import decollate_batch, DataLoader
from monai.metrics import ROCAUCMetric
from monai.networks.nets import DenseNet121
from monai.transforms import (
    Activations,
    EnsureChannelFirst,
    AsDiscrete,
    Compose,
    LoadImage,
    RandFlip,
    RandRotate,
    RandZoom,
    ScaleIntensity,
)
from monai.utils import set_determinism

print_config()

import pandas as pd
import pydicom
from torchvision import transforms
import pickle
import cv2

from sklearn.model_selection import train_test_split

# Timing utility
from timeit import default_timer as timer
from tqdm import tqdm

import torch
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score,
recall_score, f1_score, roc_curve, auc
from sklearn.metrics import roc_auc_score
from sklearn.metrics import multilabel_confusion_matrix
import seaborn as sn

import warnings
warnings.filterwarnings('ignore', category=FutureWarning)
warnings.filterwarnings('ignore', category=UserWarning)


print("all imported")

set_determinism(seed=0)

diseases = ['Aortic enlargement',
 'Cardiomegaly',
 'ILD',
 'Infiltration',
 'Lung Opacity',
 'No finding',
 'Nodule/Mass',
 'Other lesion',
 'Pleural effusion',
 'Pleural thickening',
```

```
 'Pulmonary fibrosis']

# decided on the basis of frequency of occurence of individual diseases in
images.

# Drop columns not in the list
columns_to_keep = diseases.copy()
columns_to_keep.append('image_id')

print(diseases)
print(columns_to_keep)

def delete_columns(df, columns_to_keep=columns_to_keep,
root_folder='/train'):
    columns_to_drop = [col for col in df.columns if col not in col-
umns_to_keep]
    print("COLUMNS to drop:", columns_to_drop)
    df.drop(columns=columns_to_drop, inplace=True)
    print(f"Deleted columns from {root_folder.split('/')[-1]} folder")
    return df

def remove_all_zeros(df, diseases=diseases, root_folder='/train'):
    # Remove rows where all values are 0 in the disease labels
    df = df[(df[diseases] != 0).any(axis=1)]
    df.reset_index(drop=True, inplace=True)
    return df

def add_file_path_column(df, root_folder='/train'):
    df['file_path'] = df['image_id'].apply(lambda x:
os.path.join(root_folder, f"{x}.png")) # adjust dicom or png
    print(f"Added file_path column to {root_folder.split('/')[-1]} folder")
    return df

import pandas as pd

# Load the original CSV file
#train_data = pd.read_csv("/content/vinbigdata/train.csv")
train_data = filtered_data_train

# Extract unique pairs of class_id and class_name
class_mapping = train_data[['class_id', 'class_name']].drop_duplicates()

# Sort the mapping by class_id
class_mapping = class_mapping.sort_values(by='class_id')

# Display the sorted mapping without the index
print(class_mapping.to_string(index=False))

disease_labels = class_mapping['class_name'].values
print("Disease LABELS :", disease_labels)
print('-'*100)

print("Now converting into one hot vector format")
### Convert the csv into image_id -> 0,0,0, ..... 1., 0, 1 (15 labels for-
mat)

# Convert class_id to string to enable one-hot encoding
train_data['class_id'] = train_data['class_id'].astype(str)

# Perform one-hot encoding to convert class IDs into binary columns
one_hot_encoded = pd.get_dummies(train_data['class_id'])
```

```
# Convert boolean values to integers (0 for False, 1 for True)
one_hot_encoded = one_hot_encoded.astype(int)

# Concatenate one-hot encoded columns with original DataFrame
train_data = pd.concat([train_data, one_hot_encoded], axis=1)
# print(train_data.columns)

# Group by image ID and aggregate the one-hot encoded class columns for
each class separately
train_data = train_data.groupby('image_id').agg({
#     'width': 'first',
#     'height': 'first',
    '0': 'max', '3': 'max',
    '5': 'max', '6': 'max', '7': 'max', '8': 'max', '9': 'max',
    '10': 'max', '11': 'max', '13': 'max', '14': 'max'
}).reset_index()



print("UNIQUE images IN THE DATASET:", len(train_data.image_id.unique()))
print("num rows in dataset  :", len(train_data))
# train_data.head()

# Count occurrences of each class
class_counts = train_data.iloc[:, 1:].sum()
print("TOTAL Individual class counts:-\n", class_counts)
print('-'*100)

# Rename the one-hot encoded columns
train_data = train_data.rename(columns=dict(zip(train_data.columns[1:],
disease_labels)))
print("Renamed the columns from numbers to disease_label names")

### Removing the 'no finding label' ###
train_data = delete_columns(train_data, columns_to_keep=columns_to_keep,
root_folder='train')
train_data = remove_all_zeros(train_data, diseases=diseases,
root_folder='train')


## Adding file_path column
train_data = add_file_path_column(train_data, root_folder='/content/vin-
bigdata/train')

# Split 10% of the training data as validation data, with random_state=42,
for reproducibility
train_data, val_data = train_test_split(train_data, test_size=0.2, ran-
dom_state=42)
val_data, test_data = train_test_split(val_data, test_size=0.5,ran-
dom_state=42)

# Resetting the index, very important
train_data.reset_index(drop=True, inplace=True)
val_data.reset_index(drop=True, inplace=True)
test_data.reset_index(drop=True, inplace=True)

#adjust diseases or disease_labels
train_paths = train_data['file_path'].values
train_labels = train_data[diseases].values

val_paths = val_data['file_path'].values
val_labels = val_data[diseases].values
```

```
test_paths = test_data['file_path'].values
test_labels = test_data[diseases].values



print("length of train:", len(train_data), len(train_paths), len(train_la-
bels))
print("length of val:", len(val_data), len(val_data), len(val_data))
print("length of test:", len(test_data), len(test_data), len(test_data))


print("Unique image_ids in train:", len(train_data['image_id'].unique()))
print("Unique image_ids in val:", len(val_data['image_id'].unique()))
print("Unique image_ids in test:", len(test_data['image_id'].unique()))

"""Добавление весов классам"""

from sklearn.utils.class_weight import compute_class_weight

# Получаем уникальные классы
classes = filtered_data_train['class_name'].unique()

# Рассчитываем веса классов
class_weights = compute_class_weight(class_weight='balanced', classes=clas-
ses, y=filtered_data_train['class_name'])

# Создаем словарь с весами классов
class_weight_dict = {cls: weight for cls, weight in zip(classes,
class_weights)}

# Определяем порядок классов на основе id
class_order = ['Aortic enlargement', 'Cardiomegaly', 'ILD', 'Infiltration',
'Lung Opacity', 'Nodule/Mass', 'Other lesion', 'Pleural effusion', 'Pleural
thickening', 'Pulmonary fibrosis', 'No finding']

# Создаем отсортированный словарь с весами классов
sorted_class_weights = {class_name: class_weight_dict[class_name] for
class_name in class_order if class_name in class_weight_dict}

# Выводим отсортированный словарь
print(sorted_class_weights)

# Count occurrences of each class
class_counts = train_data.iloc[:, 1:-1].sum()
print("TRAIN Individual class counts:-\n", class_counts)

# Count occurrences of each class
class_counts = val_data.iloc[:, 1:-1].sum()
print("VAL Individual class counts:-\n", class_counts)

# Count occurrences of each class
class_counts = test_data.iloc[:, 1:-1].sum()
print("TEST Individual class counts:-\n", class_counts)

train_data.head()

val_data.head()

test_data.head()

train_image_files = [
    os.path.join('/', train_paths[i]) for i in range(len(train_paths))
```

```
]
print(len(train_image_files))

val_image_files = [
    os.path.join('/', val_paths[i]) for i in range(len(val_paths))
]
print(len(val_image_files))

test_image_files = [
    os.path.join('/', test_paths[i]) for i in range(len(test_paths))
]
print(len(test_image_files))

# Commented out IPython magic to ensure Python compatibility.
import numpy as np
import pydicom
from pydicom.pixel_data_handlers.util import apply_voi_lut

import matplotlib.pyplot as plt
# %matplotlib inline


def read_xray(path, voi_lut = True, fix_monochrome = True, ap-
ply_clahe=True, clipLimit=2.0, tileGridSize=(8,8)):
    dicom = pydicom.read_file(path)

    # VOI LUT (if available by DICOM device) is used to transform raw DICOM
data to "human-friendly" view
    if voi_lut:
        data = apply_voi_lut(dicom.pixel_array, dicom)
    else:
        data = dicom.pixel_array

    # depending on this value, X-ray may look inverted - fix that:
    if fix_monochrome and dicom.PhotometricInterpretation == "MONOCHROME1":
        data = np.amax(data) - data

    data = data - np.min(data)
    data = data / np.max(data)

    if apply_clahe:
        data = apply_clahe_to_image(data, clipLimit=clipLimit,
tileGridSize=tileGridSize)

    return data

def apply_clahe_to_image(image, clipLimit=2.0, tileGridSize=(8,8)):
    # Convert image to uint16
    image = (image * 65535).astype(np.uint16)

    # Create a CLAHE object
    clahe = cv2.createCLAHE(clipLimit=clipLimit, tileGridSize=tileGridSize)

    # Apply CLAHE
    clahe_image = clahe.apply(image)

    # Convert image back to float32 in range [0, 1]
    clahe_image = clahe_image.astype(np.float32) / 65535.0

    return clahe_image

from PIL import Image
```

```
def read_png(path):
    image = Image.open(path)
    return image

def plot_images(images, titles, rows, cols):
    fig, axes = plt.subplots(rows, cols, figsize=(9, 9))
    for i, (image, title) in enumerate(zip(images, titles)):
        ax = axes[i // cols, i % cols]
        ax.imshow(image, cmap='gray')
        ax.set_title(title)
        ax.axis('off')
    plt.show()


path = '/content/vinbigdata/train/0005e8e3701dfb1dd93d53e2ff537b6e.png'
clipLimits = [2.0, 3.0]
tileGridSizes = [(6,6),(8,8)]
images = []
titles = []


for clipLimit in clipLimits:
    for tileSize in tileGridSizes:
        images.append(read_png(path))
        titles.append(f"clipLimit={clipLimit}, tileGridSize={tileSize}")

titles = [f"CL={clipLimit}, GS={tileSize}" for clipLimit in clipLimits for
tileSize in tileGridSizes]
plot_images(images, titles, len(clipLimits), len(tileGridSizes))

import cv2

def read_png(path, apply_clahe=False, clipLimit=2.0, tileGridSize=(8, 8)):
    image = Image.open(path).convert('L')  # Загрузка изображения в оттен-
ках серого
    if apply_clahe:
        img_array = np.array(image)
        clahe = cv2.createCLAHE(clipLimit=clipLimit,
tileGridSize=tileGridSize)
        img_clahe = clahe.apply(img_array)
        image = Image.fromarray(img_clahe)
    return image

img = read_png(path, apply_clahe=True, clipLimit=2.0, tileGridSize=(8, 8))
plt.figure(figsize=(8, 8))
plt.imshow(img, cmap='gray')
plt.show()

from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

# Assuming train_image_files is a list of PNG file paths
# Example:
# train_image_files = ["path/to/png/file1.png", "path/to/png/file2.png",
...]

for i, k in enumerate(np.random.randint(len(train_image_files), size=9)):
    img_path = train_image_files[k]
    img = Image.open(img_path)
    img_gray = img.convert('L')  # Convert to grayscale if needed
    img_array = np.array(img_gray)
```

```
    print("image_shape:", img_array.shape)

    # Visualization using Matplotlib
    plt.subplot(3, 3, i + 1)
    plt.imshow(img_array, cmap='gray')
    plt.title(f"Image {i + 1}")
    plt.axis('off')

plt.show()

BATCH_SIZE = 8
IMAGE_SIZE = (224,224) # the size to which the image would be resized be-
fore passing to the model
# num_classes = len(diseases)
num_classes = len(train_data.columns) - 2
## ADJUST num_classes accordingly

print("BATCH_SIZE: ", BATCH_SIZE)
print("IMAGE_SIZE: ", IMAGE_SIZE)
print("num_classes: ", num_classes)

from monai.transforms import Compose, Lambda, EnsureChannelFirst, Sca-
leIntensity,\
 RandRotate, RandFlip, RandZoom, RandSpatialCrop, RandRotate90, ResizeWith-
PadOrCrop, Resize

def load_tensor(data):
    # Convert the data to a supported data type (e.g., float32)
    return data.astype(np.float32)


train_transforms = Compose(
    [
#         LoadImage(image_only=True),
        Lambda(load_tensor),
        EnsureChannelFirst(channel_dim=0),
        #RandSpatialCrop(IMAGE_SIZE[0],IMAGE_SIZE[1]), random_size=False),
# crops randomly ; loss of information
        Resize(spatial_size=(IMAGE_SIZE[0],IMAGE_SIZE[1])),
        #ResizeWithPadOrCrop(spatial_size=IMAGE_SIZE), # this will cen-
trally crop : loss of information
        RandRotate90(prob=0.5, spatial_axes=(0, 1)),
        ScaleIntensity(),
        RandRotate(range_x=np.pi / 12, prob=0.5, keep_size=True),
        RandFlip(spatial_axis=0, prob=0.5),
        RandZoom(min_zoom=0.9, max_zoom=1.1, prob=0.5),
    ]
)

val_transforms = Compose(
    [
#         LoadImage(image_only=True),
        Lambda(load_tensor),
        EnsureChannelFirst(channel_dim=0),
#         ResizeWithPadOrCrop(spatial_size=IMAGE_SIZE), # this will cen-
trally crop : loss of information
        Resize(spatial_size=(IMAGE_SIZE[0],IMAGE_SIZE[1])),

        ScaleIntensity(),

    ]
)
```

```
# y_pred_trans = Compose([Activations(softmax=True)])
# y_trans = Compose([AsDiscrete(to_onehot=num_class)])

y_pred_trans = Compose([Activations(sigmoid=True)]) # for multi-label
classfication
y_trans = Compose([AsDiscrete(threshold_values=True)])

class VINDR_BigData_Dataset(torch.utils.data.Dataset):
    def __init__(self, paths, labels, in_chans=1, transforms=None):
        self.paths = paths
        self.labels = labels
        self.transforms = transforms
        self.target_size = IMAGE_SIZE
        self.in_chans = in_chans

    def __len__(self):
        return len(self.paths)

    def __getitem__(self, index):
        img_path = self.paths[index]
        labels = self.labels[index]

        image = Image.open(img_path)
        image_gray = image.convert('L')  # Convert to grayscale if needed
        image_array = np.array(image_gray)

        # Expand the dimensions to add a channel dimension
        image_array = np.expand_dims(image_array, axis=0)

        if self.transforms:
            image_array = self.transforms(image_array)

        return image_array, labels


train_ds = VINDR_BigData_Dataset(train_paths, train_labels, in_chans=1,
transforms=train_transforms)
train_loader = DataLoader(train_ds, batch_size=BATCH_SIZE, shuffle=True,
num_workers=4)

val_ds = VINDR_BigData_Dataset(val_paths, val_labels, in_chans=1, trans-
forms=val_transforms)
val_loader = DataLoader(val_ds, batch_size=BATCH_SIZE, num_workers=4)
test_ds = VINDR_BigData_Dataset(test_paths, test_labels, in_chans=1, trans-
forms=val_transforms)
test_loader = DataLoader(test_ds, batch_size=BATCH_SIZE, num_workers=4)

print("No. of TRAIN batches:", len(train_loader))
print("No. of VAL batches:", len(val_loader))
print("No. of TEST batches:", len(test_loader))

from thop import profile
from thop import clever_format
import torch

def display_params_flops(model):
    #params

    num_params = sum(p.numel() for p in model.parameters())
    num_params_millions = num_params / 1e6
    print(f"Number of parameters in millions: {num_params_millions:.2f} M")
```

```
    num_params = sum(p.numel() for p in model.parameters() if p.re-
quires_grad)
    num_params_millions = num_params / 1e6
    print(f"Number of trainable parameters in millions: {num_params_mil-
lions:.2f} M")

    #FLOPS
    input_size = (1, 1, IMAGE_SIZE[0], IMAGE_SIZE[1])

    # Move the model to GPU if available
    if torch.cuda.is_available():
        model = model.cuda()

    # Use thop.profile to count FLOPs
    input_tensor = torch.randn(*input_size)
    if torch.cuda.is_available():
        input_tensor = input_tensor.cuda()
    flops, params = profile(model, inputs=(input_tensor,))

    # Convert FLOPs to gigaFLOPs and format the results
    flops, params = clever_format([flops, params], "%.2f")
    print(f"FLOPs: {flops}, Params: {params}")

pip install timm

import torch
import torch.nn as nn
import torchvision.models as models

from torch import nn
from timm import create_model


class SwinTransformerModel(nn.Module):
    def __init__(self, num_classes, fine_tune=False):
        super(SwinTransformerModel, self).__init__()
        self.swin = create_model(
            'swin_large_patch4_window7_224.ms_in22k',
            pretrained=True,
            num_classes=num_classes,
            in_chans=1
        )
        if not fine_tune:
            for param in self.swin.parameters():
                param.requires_grad = False

            for param in self.swin.head.parameters():
                param.requires_grad = True

    def forward(self, x):
        x = self.swin(x)
        return x

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = SwinTransformerModel(num_classes, fine_tune=False)
model.to(device)

# print(model)
print()

x = torch.randn(1, 1, IMAGE_SIZE[0], IMAGE_SIZE[1]).to(device)
output = model(x)
```

```
print("Model output's shape:", output.shape)
print(output) # logits
display_params_flops(model)

import torch.nn as nn
model = SwinTransformerModel(num_classes, fine_tune=False) # change
fine_tune as required
model.to(device)

loss_function = nn.BCEWithLogitsLoss(pos_weight=torch.ten-
sor(list(sorted_class_weights.values()))).to(device)
optimizer = torch.optim.Adam(model.parameters(), 1e-5) # adjust learning
rate
```