МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра «Прикладная математика и информатика»
(наименование)

01.03.02 Прикладная математика и информатика
(код и наименование направления подготовки, специальности)

Компьютерные технологии и математическое моделирование
(направленность (профиль)/специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Алгоритмы анализа и классификации патологий на медицинских данных»

Обучающийся Н.А. Григорьев _____
(Инициалы Фамилия) (личная подпись)

Руководитель М.Г. Лисовская
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант к.ф.н., доцент М.В. Дайнеко
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2025

Аннотация

Тема выпускной квалификационной работы: «Алгоритмы анализа и классификации патологий на медицинских данных».

Целью работы является классификация патологий на рентгенографическом снимке грудной клетки с минимальной погрешностью.

Объектом исследования выступает процесс классификации патологий на медицинских данных.

Предметом исследования являются алгоритмы анализа и классификации патологий на медицинских данных.

Данная выпускная квалификационная работа состоит из введения, трех разделов, заключения и списка используемой литературы.

Во введении обосновывается актуальность темы, формулируются цель и задачи исследования.

В первом разделе рассматривается предметная область, и анализируются данные о заболеваемости, а также обсуждается важность диагностики заболеваний органов дыхания.

Во втором разделе представлен обзор методов сегментации медицинских изображений и разработан гибридный алгоритм для улучшения качества анализа рентгенограмм.

В третьем разделе описывается реализация нейронной сети для классификации патологий, проведено функциональное тестирование и анализ результатов работы.

В заключении подводятся итоги исследования и обсуждаются перспективы дальнейшего развития системы.

Работа включает 84 страницы текста, 40 рисунков, 7 таблиц, 29 формул, 24 источника и 2 приложения.

Abstract

The title of the graduation work is «Algorithms of analysis and classification of pathologies based on medical data».

This research is devoted to developing and implementing the effective methods of diagnosing respiratory diseases using the modern machine learning algorithms.

The aim of this graduation work is classify the pathologies on chest X-ray images with a minimal error. To achieve this aim, it is necessary to solve the tasks related to analyzing the image segmentation methods, developing a hybrid algorithm for effective segmentation, selecting and implementing the pathology classification algorithm, as well as testing its effectiveness.

The object of the research is the process of classifying the pathologies based on medical data. The subject of the graduation work is the algorithms of pathologies analysis and classification based on medical data.

This graduation work consists of an introduction, 3 parts, 40 figures, 7 tables, a conclusion, a list of 24 references, and 2 appendices.

The introduction justifies the relevance of the topic, as well as formulates the aim and the tasks of the research.

The first part examines the subject area, analyzes the morbidity data, and discusses the importance of diagnosing respiratory diseases.

The second part gives an overview of the medical image segmentation methods and presents the hybrid algorithm designed to improve the quality of X-ray analysis.

The third part reveals the implementation of pathologies classification system, deals with functional testing, and carries out an analysis of the algorithm's performance results.

In conclusion, it should be noted that the developed system of analyzing and classifying the pathologies in chest X-ray images has achieved a classification accuracy of approximately 94% thanks to the use of the hybrid segmentation method and a convolutional neural network based on VGG19. We are planning to integrate the system with the other analytical methods to enhance the diagnostic accuracy.

Содержание

Введение

В структуре общей заболеваемости населения России болезни органов дыхания занимают первое место, составляя около 25-27 процентов всех зарегистрированных случаев в период с 2020 по 2022 годы [7].

В официальном сборнике Росстата «Здравоохранение в России – 2023» данные о болезнях органов дыхания можно систематизировать следующим образом:

Общая заболеваемость болезнями органов дыхания: в 2022 году уровень общей заболеваемости болезнями органов дыхания составил около 470 случаев на 1000 человек населения, что выше показателей предыдущих лет (415 в 2020 году, 379 в 2010 году).

Первичная заболеваемость: в 2022 году первичная заболеваемость болезнями органов дыхания достигла 422 случаев на 1000 человек (по сравнению с 367 в 2020 году).

Болезни органов дыхания занимают 6-е место среди причин смертности в РФ. Кроме того заболевания органов дыхания остаются лидирующей причиной временной нетрудоспособности.

Согласно данным Eurostat, в 2021 в Европейском Союзе зарегистрировано 324 300 смертей от заболеваний дыхательной системы, что составляет 6.1% от общего числа летальных исходов. Пневмония является одной из основных причин смертности в этой категории [14].

Учитывая высокую заболеваемость и смертность от заболеваний органов дыхания, в частности пневмонии, становится очевидной необходимость разработки эффективных методов диагностики и классификации этих патологий.

При диагностике заболеваний органов дыхания рентгенография играет ключевую роль. Однако, несмотря на наличие квалифицированных врачей, процесс интерпретации рентгенограмм может быть субъективным и зависеть от опыта специалиста. В этом контексте алгоритмы классификации, такие

как сверточные нейронные сети, становятся важным инструментом для повышения точности и объективности диагностики.

Алгоритмы классификации позволяют автоматизировать процесс анализа рентгенограмм, что снижает вероятность человеческой ошибки и ускоряет диагностику. Они обучаются на больших объемах данных, что позволяет им выявлять паттерны и аномалии, которые могут быть неочевидны для глаза врача. Это особенно важно в условиях высокой заболеваемости, когда количество пациентов может превышать возможности врачей.

Для эффективной классификации патологий на медицинских изображениях, таких как рентгеновские снимки, необходимо выделить области интереса. В этом контексте сегментация играет ключевую роль, так как она позволяет выделить определённые области на изображении, которые могут быть интересны для дальнейшего анализа. Она помогает сосредоточиться на тех областях, которые действительно требуют внимания, что в свою очередь повышает точность диагностики. Кроме того, сегментация помогает избежать захвата лишней информации, которая может привести к неправильной интерпретации при классификации патологий.

Актуальность данной бакалаврской работы обусловлена высоким уровнем заболеваемости и смертности от заболеваний органов дыхания, особенно в условиях пандемии и увеличения числа респираторных инфекций. Разработка и внедрение эффективных методов диагностики, основанных на современных алгоритмах машинного обучения, могут значительно улучшить качество медицинской помощи.

Объектом исследования выступает процесс классификации патологий на медицинских данных.

Предметом исследования являются алгоритмы анализа и классификации патологий на медицинских данных.

Целью работы является классификация патологий на рентгенографическом снимке грудной клетки с минимальной погрешностью.

Для достижения выше поставленной цели, необходимо решить следующие задачи:

- проанализировать существующие методы сегментации изображений, которые позволяют улучшить качество анализа медицинских данных;
- разработать и реализовать гибридный алгоритм для эффективной сегментации рентгенографических снимков грудной клетки;
- проанализировать существующие алгоритмы классификации патологий на медицинских данных;
- выбрать и реализовать наиболее подходящий алгоритм для решения поставленной задачи;
- провести тестирование и сделать выводы о качестве классификации патологий.

Данная работа содержит: введение, три раздела и заключение.

Первый раздел работы посвящен анализу методов сегментации и описанию видов патологий на рентгенографических снимках. Рассматривается рост заболеваемости пневмонией и актуальность проблемы.

Во втором разделе описывается задача классификации медицинских изображений. Рассматриваются реализации различных методов сегментации, на основе которых разрабатывается гибридный алгоритм сегментации медицинских изображений.

В третьем разделе рассматривается реализация сверточной нейронной сети для классификации патологий на основе модели VGG19. Также описывается реализация и оценка гибридного алгоритма сегментации медицинских изображений, включая обработку DICOM-файлов и создание графического интерфейса пользователя.

Результатом работы является система для анализа и классификации патологий на медицинских данных, которая улучшает качество диагностики и способствует более эффективному оказанию медицинской помощи.

1 Описание методов анализа медицинских изображений

1.1 Определение патологии органов дыхания для исследования

Заболевания органов дыхания представляют собой обширную группу патологий, среди которых можно выделить основные, такие как:

- пневмония: воспаление легких, вызванное инфекциями (бактериальными, вирусными или грибковыми). Симптомы включают кашель, одышку, боль в груди и лихорадку;
- бронхит: воспаление бронхов, которое может быть острым или хроническим. Характеризуется кашлем, выделением мокроты и затруднением дыхания;
- хроническая обструктивная болезнь легких (ХОБЛ): хроническое заболевание, включающее хронический бронхит и эмфизему, часто вызванное курением;
- астма: хроническое воспалительное заболевание дыхательных путей, вызывающее приступы одышки и кашля;
- туберкулез: инфекционное заболевание, вызываемое бактерией «Mycobacterium tuberculosis», которое может поражать легкие и другие органы.

Среди этих заболеваний пневмония выделяется как наиболее распространенная. В последние годы наблюдается значительный рост заболеваемости этой болезни.

Согласно данным Роспотребнадзора, осенью 2024 года зафиксирован резкий рост заболеваемости пневмонией в 16 регионах России. В некоторых из них уровень заболеваемости в 3 раза превышает среднероссийский уровень.

Учитывая высокую заболеваемость пневмонией и актуальность проблемы, выбор данного заболевания для классификации является

обоснованным и подчеркивает значимость пневмонии для дальнейшего изучения.

1.2 Методы сегментации медицинских изображений

Сегментация медицинских изображений является важной задачей в области компьютерной медицины, которая способствует более точному анализу различных заболеваний, включая заболевания легких. Она позволяет выделять области на изображении, подверженные заболеваниям, что улучшает визуализацию патологических очагов и повышает качество классификации.

Сегментация будет осуществляться на полутоновых изображениях, что требует применения специфических методов обработки изображений, способных эффективно выделять контуры легких.

Существуют различные методы, которые могут быть использованы для решения задач сегментации медицинских изображений. К ним относятся:

- методы на основе пороговой обработки: эти методы основываются на установлении пороговых значений для разделения объектов на изображении. Например, можно использовать глобальные или адаптивные пороги для выделения легких на рентгенограммах. Однако такие методы могут быть чувствительны к шуму и изменению освещения;
- методы, основанные на контурном анализе: эти методы используют алгоритмы для выявления границ объектов. Одним из популярных подходов является использование операторов градиента, таких как оператор Собеля или оператор Кэнни, для выделения контуров легких. Тем не менее, эти методы могут не всегда обеспечивать высокую точность в сложных случаях, когда контуры легких плохо выражены.

– методы, основанные на области: эти методы фокусируются на выделении областей (водораздел), которые имеют схожие характеристики. Например, алгоритмы растягивания области (region growing) могут быть использованы для сегментации легких, начиная с инициализации области в пределах легких и последующего расширения на основе критериев схожести.
– методы на основе кластеризации: эти методы используют алгоритмы кластеризации (K-Means) для группировки пикселей в кластеры схожих свойств.

Каждый из этих методов имеет свои преимущества и недостатки, и выбор подхода зависит от конкретной задачи, качества исходных изображений и требований к точности сегментации. В данной работе будет проведен анализ существующих методов и предложены новые подходы, направленные на улучшение сегментации легких на рентгенограммах, что, в свою очередь, может способствовать более ранней и точной диагностике патологий.

1.3 Обзор существующих методов сегментации для задачи анализа медицинских изображений

Рассмотрим четыре метода сегментации: пороговая сегментация, K-Means, оператор Собеля и водораздел.

1.3.1 Пороговая сегментация

Пороговая сегментация – это метод, который разделяет изображение на области на основе заданного порога интенсивности пикселей.

Ключевые шаги метода:

- Определение порога: пусть I(x,y) – это функция интенсивности изображения, где (x,y) – координаты пикселя. Пороговая сегментация включает в себя выбор порогового значения T.
- Классификация пикселей: каждый пиксель изображения классифицируется (1) по следующему правилу:
    а) 1 соответствует объекту (легким человека);
    б) 0 – фону.
- Выбор порога: порог T может быть выбран вручную или автоматически. Один из популярных методов автоматического выбора порога – метод Отцу.
- Постобработка: после применения пороговой сегментации могут быть использованы морфологические операции для улучшения качества сегментации, удаления шумов и заполнения пробелов.

Формула (1) иллюстрирует классификацию пикселей при применении пороговой сегментации.

$$S(x, y) = \begin{cases} 1, & если\ I(x, y) \geq T \\ 0, & если\ I(x, y) < T \end{cases}, \quad (1)$$

где S(x, y) – бинарное изображение.

Плюсы пороговой сегментации:

- простота и скорость выполнения;
- легкость в реализации;
- хорошо работает на изображениях с четкими границами объектов.

Минусы пороговой сегментации:

- метод неэффективен при наличии шумов или неоднородного освещения;

– необходимость выбора оптимального порога, что может быть сложной задачей.

Метод может быть улучшен с помощью адаптивной пороговой сегментации, которая учитывает локальные характеристики изображения.

1.3.2 Метод K-Means

K-Means – это алгоритм кластеризации, который делит данные на k кластеров, минимизируя внутрикластерные расстояния.

Основные этапы алгоритма:

– Инициализация: необходимо выбрать количество кластеров k. Инициализировать центры кластеров случайным образом или с помощью определенных методов;
– Обработка изображения: изображение преобразуется в одномерный массив пикселей. Каждый пиксель представлен как вектор, где значение пикселя является единственным элементом;
– Кластеризация: для каждого пикселя вычисляется расстояние до центров кластеров. Расстояние между пикселями и центрами кластеров обычно измеряется с использованием евклидового расстояния (2);
– Обновление меток кластеров: каждому пикселю присваивается метка кластера, к которому он ближе всего.

Евклидово расстояние, представленное в формуле (2).

$$d_2(x, y) = \sqrt{\sum_{i=1}^{m} (x - y)^2} \quad (2)$$

Плюсы алгоритма K-Means:

– простота реализации и понимания;
– быстрая работа на больших наборах данных;

– хорошо работает, если кластеры имеют сферическую форму.

Минусы алгоритма K-Means:

– необходимость заранее задавать количество кластеров (k);
– чувствительность к выбросам и шуму;
– не всегда находит глобальный минимум (может застрять в локальных минимумах).

1.3.3 Оператор Собеля

Оператор Собеля – это метод выделения границ, который использует градиенты для нахождения изменений интенсивности в изображении.

Основные этапы метода:

– Градиент изображения: градиент изображения I(x, y) представляет собой вектор, который указывает направление и величину изменения интенсивности в каждой точке изображения;
– Ядра Собеля: оператор Собеля использует два свёрточных ядра для вычисления градиента: одно для горизонтального направления (3), а другое – для вертикального (4);
– Вычисление градиента: для вычисления градиента изображения, оператор Собеля свёртывает исходное изображение I(x, y) с ядрами G_x и G_y (5) и (6);
– Величина и направление градиента: после вычисления градиентов по обоим направлениям, можно получить величину градиента G и его направление θ (8);
– Пороговая обработка: для выделения границ в изображении часто применяется пороговая обработка к величине градиента G. Пиксели, у которых величина градиента превышает заданный порог, считаются границами.

Горизонтальное ядро G_y:

$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad (3)$$

Вертикальное ядро G_x:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad (4)$$

Свёртка оператором Собеля исходного изображения I(x, y) с ядром G_x:

$$G_x(x, y) = I(x, y) * G_x \quad (5)$$

,

где * – операция свёртки.

Свёртка оператором Собеля исходного изображения I(x, y) с ядром G_y:

$$G_y(x, y) = I(x, y) * G_y \quad (6)$$

Плюсы:

- эффективен для выделения границ объектов;
- простота реализации и быстрая работа.

Минусы:

- чувствительность к шуму;
- может не обнаруживать тонкие границы;
- часто используется в качестве предварительного этапа перед другими методами сегментации для улучшения качества границ.

1.3.4 Метод водораздела

Метод водораздела основан на концепции топологии, где изображение рассматривается как рельефная поверхность, и сегментация осуществляется путем нахождения «водоразделов» между различными областями.

Основные этапы метода:

- Градиентное изображение: начинаем с вычисления градиента изображения I(x, y), который показывает, как изменяется интенсивность в каждом пикселе. Градиент можно вычислить с помощью оператора Собеля (7);
- Топографическая карта: градиентное изображение интерпретируется как топографическая карта, где высокие значения градиента представляют собой «горы» (объекты), а низкие значения – «долины» (фон);
- Определение водоразделов: водоразделы представляют собой линии, которые разделяют разные «горы». Для нахождения водоразделов используется концепция маркерной сегментации;
- Определяются маркеры (начальные точки) для объектов, которые нужно выделить. Эти маркеры могут быть выбраны вручную или автоматически;
- Затем применяется алгоритм, который разливает воду из этих маркеров, заполняя «долины» и останавливаясь на «горах»;
- Алгоритм сегментации: алгоритм водораздела можно описать следующим образом:
    а) инициализируем маркеры M(x, y) для каждого объекта;
    б) для каждого пикселя (x, y) в изображении:
        1) если M(x, y) соответствует маркеру, продолжаем разливать воду;
        2) если достигаем границы (горы), останавливаемся и фиксируем сегментацию.

Вычислить градиент с помощью оператора Собеля можно по формуле (7).

$$G(x, y) = \sqrt{G_x^2(x, y) + G_y^2(x, y)} \quad (7)$$

Направление градиента вычисляется по формуле (8).

$$\theta = arctan\left(\frac{G_y(x, y)}{G_x(x, y)}\right), \quad (8)$$

Плюсы метода водораздела:

- эффективен для сегментации объектов с нечеткими границами;
- может учитывать сложные формы объектов.

Минусы метода водораздела:

- высокая вычислительная сложность;
- чувствительность к шуму в изображении;
- может потребовать предварительной обработки для улучшения результатов.

Часто используется в сочетании с другими методами, такими как предварительная фильтрация для уменьшения шума.

Каждый из методов имеет свои преимущества и недостатки, и выбор подходящего метода зависит от конкретной задачи, качества изображений и требований к точности сегментации. Часто для достижения наилучших результатов используются комбинации различных методов.

1.4 Анализ стандартизированного формата медицинских данных DICOM

Работа с форматом DICOM. DICOM (Digital Imaging and Communications in Medicine) – это стандарт, используемый для хранения и передачи медицинских изображений и связанных с ними данных. Он включает в себя как саму информацию об изображении, так и метаданные (рисунок 1), такие как информация о пациенте, исследовании, устройстве и т. д.

```
(0010,0010) Patient's Name                      PN: ''
(0010,0020) Patient ID                          LO: ''
(0010,0021) Issuer of Patient ID                LO: ''
(0010,0030) Patient's Birth Date                DA: ''
(0010,0040) Patient's Sex                       CS: ''
(0010,1000) Other Patient IDs                   LO: ''
(0010,1001) Other Patient Names                 PN: ''
(0010,1020) Patient's Size                      DS: None
(0010,1030) Patient's Weight                    DS: None
(0010,1040) Patient's Address                   LO: ''
(0010,2150) Country of Residence                LO: 'Не выбрано'
(0010,2152) Region of Residence                 LO: 'Не выбрано'
(0010,2154) Patient's Telephone Numbers         SH: ''
(0010,4000) Patient Comments                    LT: ''
```

Рисунок 1 – Метаданные

Формат DICOM занимает значительное количество места, происходит это по нескольким причинам:

- высокое разрешение изображений: медицинские изображения, такие как рентгеновские снимки, МРТ и КТ, часто имеют высокое разрешение и могут содержать большое количество пикселей;
- многослойные изображения: некоторые медицинские изображения, такие как МРТ и КТ, могут состоять из множества срезов, каждый из

которых представляет собой отдельное изображение. Это может значительно увеличить общий объем данных;

– метаданные: DICOM-файлы содержат не только пиксельные данные, но и обширные метаданные, которые могут включать информацию о пациенте, исследовании, устройстве, параметрах сканирования и т. д. Эти метаданные необходимы для правильной интерпретации изображений и могут занимать значительное место.

В нашем алгоритме метаданные использоваться не будут, поэтому для уменьшения объема данных, и как следствие ускорение работы алгоритма, рационально будет извлекать снимки из DICOM-файла, и сохранять их в формате PNG.

2 Формирование решения задачи классификации медицинских изображений и разработка гибридного алгоритма сегментации

2.1 Инструменты и базы данных для работы с медицинскими изображениями

В качестве языка программирования будет использоваться Python, а интегрированной средой разработки (IDE) – PyCharm.

В данной работе использовались две различные базы данных рентгенографических снимков грудной клетки. Первая база данных, доступная на платформе Kaggle, включает более 5000 рентгенографических снимков грудной клетки, на которых представлены как признаки пневмонии, так и снимки без патологий [18]. Эти изображения аннотированы, что позволяет использовать их для обучения моделей, способных различать здоровые легкие и легкие, пораженные пневмонией. Вторая база данных включает рентгенографические снимки грудной клетки, предоставленные Национальными институтами здравоохранения (NIH). Эта база данных содержит более 100 000 изображений, которые охватывают широкий спектр заболеваний легких, включая пневмонию, туберкулез, рак легких и другие патологии [19]. Снимки в этой базе данных также аннотированы, что способствуют проведению анализа и разработке алгоритмов для автоматической диагностики.

Основной исходный снимок для тестирования различных алгоритмов:

Рисунок 2 – Исходный снимок

2.2 Реализация существующих методов сегментации

Пороговая сегментация

Для начала реализуем простой алгоритм пороговой сегментации. Процесс начинается с загрузки изображения с использованием функции «cv2.imread()». Путь к изображению указывается в виде строки, а флаг «cv2.IMREAD_GRAYSCALE» указывает на необходимость загрузки изображения в градациях серого.

После загрузки изображения применяется пороговая обработка для получения бинарного изображения. Для этого используется функция «cv2.threshold()», которая принимает следующие параметры:

1) image: изображение в градациях серого, загруженное в предыдущем этапе.

2) 120: пороговое значение, выбранное для сегментации. Пиксели, значения которых меньше 120, будут установлены в максимальное значение

(255), а пиксели, значения которых больше или равны 120, будут установлены в 0. Выбор порогового значения 120 будет обоснован в дальнейшем.

3) 255: максимальное значение, которое будет присвоено пикселям, прошедшим порог.

Для визуализации полученного бинарного изображения используется библиотека «matplotlib», что позволяет наглядно оценить результаты сегментации (рисунок 3).

Рисунок 3 – Бинарное изображение

Получили бинарное изображение. Пока довольно тяжело оценить, насколько эффективно работает алгоритм. Перейдем к следующему этапу – созданию маски и наложению её на исходное изображение.

В процессе реализации были внесены следующие изменения в код:

Для удаления шума и заливки областей используется функция «remove_small_objects» из библиотеки «skimage.morphology», которая удаляет небольшие объекты из бинарного изображения. Параметр «min_size=5000» указывает, что объекты с площадью менее 5000 пикселей будут удалены. Сначала бинарное изображение преобразуется в логический

массив, после чего результат конвертируется обратно в тип «uint8» и умножается на 255 для получения бинарного изображения. Далее создается структурный элемент в форме эллипса размером 5x5 пикселей с помощью функции «cv2.getStructuringElement()». Затем применяется операция морфологического закрытия, которая помогает заполнить небольшие отверстия в объектах и соединить близкие объекты.

Для нахождения связных областей и их сортировки по площади используется функция «measure.label()» из библиотеки «skimage», которая маркирует связные области в бинарном изображении. Затем с помощью «measure.regionprops()» вычисляются свойства этих областей, такие как площадь и координаты. Области сортируются по площади в порядке убывания.

Создается маска, которая будет содержать только две самые большие области. Для каждой области в списке «largest_regions» соответствующие пиксели в маске устанавливаются в 255.

Наконец, маска накладывается на исходное изображение с помощью операции побитового И (cv2.bitwise_and), что позволяет выделить только те области, которые были определены в маске.

Результат работы алгоритма:

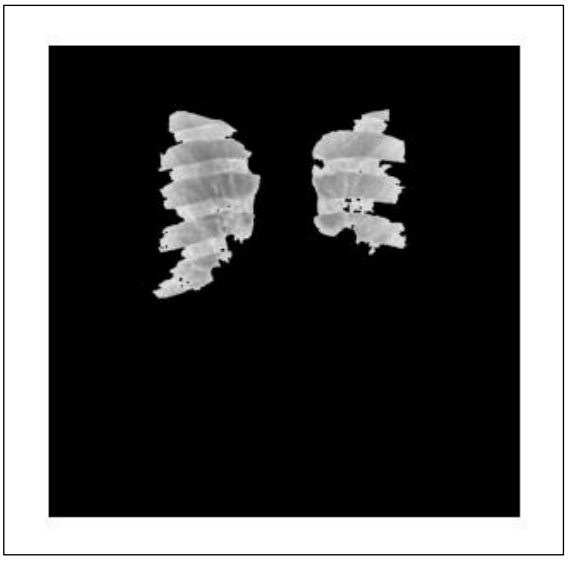

Рисунок 4 – Результат сегментации

Наложенная маска на исходное изображение позволяет визуально оценить эффективность алгоритма сегментации. Для более точной оценки результатов сегментации используются количественные метрики, такие как индекс Жаккара и коэффициент Дайса. Эти метрики были выбраны, поскольку они позволяют количественно оценить схожесть между сегментированным изображением и истинными аннотациями, что является важным для понимания качества работы алгоритма.

Индекс Жаккара (англ. Jaccard Index) – это метрика, используемая для оценки схожести между двумя наборами данных. В контексте сегментации изображений он позволяет измерить, насколько хорошо сегментированное изображение совпадает с истинной областью.

Для вычисления индекса берется маска, полученная в процессе сегментации снимка, маска представляет собой бинарное изображение, где пиксели, относящиеся к интересующей области, имеют значение 1, а остальные – 0. Затем для вычисления потребуется бинарное изображение, которое представляет собой истинные аннотации, где пиксели, относящиеся к легким, также имеют значение 1, а остальные – 0. Оба бинарных изображения необходимо преобразовать в одномерные массивы, чтобы упростить дальнейшие вычисления. Это можно сделать с помощью библиотеки NumPy (рисунок 5).

Далее находится пересечение двух массивов, определив количество пикселей, которые равны 1 в обоих массивах, что соответствует количеству истинно положительных (TP) пикселей. Затем вычисляется объединение массивов, определив количество пикселей, которые равны 1 хотя бы в одном из массивов. Это значение включает в себя истинно положительные (TP), а также количество ложно положительных (FP) и ложно отрицательных (FN) пикселей. Теперь, когда посчитаны значения пересечения и объединения, можно вычислить индекс Жаккара по формуле (9).

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}, \quad (9)$$

где $|A \cap B|$ – это количество пикселей в пересечении (TP);

$|A \cup B|$ – это количество пикселей в объединении (TP + FP + FN).

```
segmented_array = mask.flatten()

true_array = true_image.flatten() # бинарное изображение аннотаций

intersection = np.sum((segmented_array == 255) & (true_array == 255))

union = np.sum((segmented_array == 255) | (true_array == 255))

jaccard_index = intersection / union

print(f"Индекс Жаккара: {jaccard_index:.6f}")
```

Рисунок 5 – Вычисление индекса Жаккара

Индекс Жаккара принимает значения от 0 до 1. Значение 0 означает, что сегментированное изображение не пересекается с истинным изображением. Значение 1 означает, что сегментированное изображение полностью совпадает с истинным изображением.

Получаем J равный 0,150801 или около 15% совпадения сегментированного изображения с размеченной аннотацией (рисунок 6).

```
Индекс Жаккара: 0.150801
```

Рисунок 6 – Индекс Жаккара

Коэффициент Дайса (англ. Dice Coefficient), в отличие от индекса Жаккара, учитывает пересечение с удвоением, что делает его более чувствительным к небольшим изменениям в сегментации. Коэффициент вычисляется по формуле (10).

$$D(A, B) = \frac{2 \times |A \cap B|}{|A| + |B|}, \tag{10}$$

где $|A|$ – это количество элементов в множестве A (количество пикселей, равных 255 в сегментированном изображении);

$|B|$ – это количество элементов в множестве B (количество пикселей, равных 255 в аннотации).

Получаем D равный 0,262080, что соответствует примерно 26% совпадения сегментированного изображения с размеченной аннотацией (рисунок 7).

Рисунок 7 – Dice коэффициент

Результат сегментации явно не удовлетворителен. Выделена лишь малая часть области легких. Выше в тексте я упомянул, что значение порога было выставлено в 120. Мы, конечно, можем дальше подбирать значение порога вручную, например, попробовать взять значение 150 и постепенно увеличивать его, пока существенная часть легких не будет выделена. Но это абсолютно не имеет смысла, ведь если для одного рентгеновского снимка оптимальное значение порога будет 150, то это не значит, что это же значение будет подходить к другим снимкам. Конечно, можно применить предобработку изображений, включая увеличение контрастности и обрезку

снимков. Но в любом случае оптимальное значение порога будет варьироваться для каждого отдельного снимка.

Для решения данной проблемы мы применим метод, разработанный доктором инженерии Нобуюки Отсу из Токийского университета. Этот метод, названный в честь своего создателя, направлен на поиск порога, который минимизирует внутриклассовую дисперсию, что, в свою очередь, способствует более качественной сегментации данных.

Добавим одно единственное изменение в код нашей программы:

Рисунок 8 – Изменения в коде программы

Мы устанавливаем пороговое значение в 0 и используем «cv2.THRESH_OTSU». Это означает, что функция будет автоматически определять оптимальное пороговое значение на основе гистограммы изображения.

Запускаем программу. После завершения работы алгоритма, получаем следующий результат:

Рисунок 9 – Результат работы алгоритма

По итогам сегментации получаем индекс Жаккара равный 0.482879 или около 48% совпадения сегментированного изображения с размеченной аннотацией. И Dice коэффициент равный 0.651272 (рисунок 10), что соответствует примерно 65% совпадения сегментированного изображения с размеченной аннотацией.

```
Индекс Жаккара: 0.482879
Dice коэффициент: 0.651272
```

Рисунок 10 – Метрики

Теперь результат стал гораздо лучше, большая часть области легких выделена.

Для измерения времени выполнения алгоритма была применена функция «perf_counter()» из модуля «time». В ходе эксперимента было проведено десять замеров времени выполнения, результаты которых представлены в таблице 1.

Таблица 1 – Время выполнения алгоритма

Номер запуска	Время выполнения, с.
1	0.08436
2	0.08082
3	0.07502
4	0.07377
5	0.07832
6	0.07535
7	0.07501
8	0.07451
9	0.07371
10	0.08844

На основе полученных результатов из таблицы 1, среднее время выполнения алгоритма составило 0,07793 секунд.

Если требуется получить инвертированное изображение, то достаточно использовать функцию «cv2.bitwise_not()» непосредственно на исходном изображении, на которое уже наложена маска. В таком случае результат работы программы будет следующим:

Рисунок 11 – Инвертированный результат работы алгоритма

Метод K-Means

Реализуем алгоритм сегментации легких с использованием метода K-Means. Изображение загружается с помощью функции «io.imread()» и сохраняется в переменной «gray_image». Путь к изображению задается в переменной «image_path». Изображение загружается в градациях серого, так как параметр «as_gray=True» указывает на необходимость преобразования цветного изображения в одноканальное.

Для применения алгоритма K-Means изображение преобразуется в одномерный массив пикселей с помощью метода «reshape()». Это необходимо, так как K-средние работают с плоскими массивами данных.

Создается объект «KMeans» с заданным количеством кластеров, в данном случае k равное 2, затем он обучается на массиве пикселей (рисунок 12). После обучения метки кластеров извлекаются и преобразуются обратно в форму изображения.

```
k = 2
kmeans = KMeans(n_clusters=k, random_state=0)
kmeans.fit(pixel_values)
```

Рисунок 12 – Количество кластеров

Для удаления мелких объектов используется функция «morphology.remove_small_objects()», которая удаляет объекты, размер которых меньше заданного порога (min_size=5000). Результат преобразуется в бинарное изображение.

С помощью «np.bincount()» подсчитываются количество пикселей в каждом кластере. Кластер с меньшими значениями (предполагается, что это легкие) выбирается для создания маски.

Для нахождения связных областей используется функция «measure.label()», которая позволяет идентифицировать связные области в сегментированном изображении. Затем вычисляются свойства этих областей с помощью «measure.regionprops()».

Области сортируются по их площади в порядке убывания, и выбираются две самые большие области. Создается пустая маска, и для каждой из двух самых больших областей в «largest_regions» устанавливаются соответствующие пиксели в маске.

Наконец, маска накладывается на исходное изображение с помощью функции «cv2.bitwise_and()», что позволяет выделить области легких. Результаты визуализируются с использованием библиотеки «matplotlib».

Для инициализации центроидов в алгоритме K-Means используется метод k-means++, который обеспечивает равномерное распределение начальных центроидов, что ускоряет сходимость алгоритма. Вероятность выбора точки x в качестве нового центроида определяется по формуле (11).

$$P(x) = \frac{D(x)^2}{\sum_{x_i \in X} D(x_i)^2}, \tag{11}$$

где D(x) – расстояние до ближайшего существующего центроида.

Расстояние до ближайшего центроида D(x) вычисляется как минимальное евклидово расстояние между точкой x и множеством центроидов $C = \{c_1, c_2, ..., c_k\}$, по формуле (12).

$$D(x) = \min_{c \in C} ||x - c|| \tag{12}$$

Евклидово расстояние определяется как:

$$\|x - c\| = \sqrt{\sum_{i=1}^{n} (x_i - c_i)^2} \tag{13}$$

Назначение точек кластерам происходит следующим образом. Каждой точке x_i назначается кластер k с ближайшим центроидом u_k на основе квадратов евклидова расстояния, что оптимизирует вычисления. Формула назначения кластеров выглядит следующим образом:

$$c^{(i)} = arg\ \min_k ||x_i - u_k||^2 \quad , \tag{14}$$

где u_k – центроид кластера k;

$||x_i - u_k||^2$ – квадрат евклидова расстояния.

Квадрат евклидова расстояния:

$$\|x - u\|^2 = \sum_{i=1}^{n} (x_i - u_i)^2 \tag{15}$$

Обновления центроидов происходят следующим образом. Центроид u_k перемещается в центр масс кластера, пересчитываясь как среднее арифметическое всех точек кластера C_k (16).

$$u_k = \frac{1}{|C_k|} \sum_{i \in C_k} x_i \quad , \tag{16}$$

где C_k – множество точек кластера k.

Критерий останова (инерция) представляет собой сумму квадратов расстояний от каждой точки до ближайшего центроида, которая минимизируется алгоритмом. Формула инерции для критерия останова:

$$J = \sum_{i=0}^{n} \min_{u_k \in C} //x_i - u_k//^2 \tag{17}$$

Алгоритм останавливается, когда:

$$\frac{|J_t - J_{t-1}|}{J_{t-1}} < tol \ , \tag{18}$$

где J_t – инерция на итерации t.

По умолчанию tol = 1e-4, также алгоритм заканчивает работу, если достигнуто максимальное число итераций (по умолчанию max_iter = 300).

На рисунке 13 изображена маска, полученная в ходе работы алгоритма K-means. Конечный результат работы алгоритма K-Means:

Рисунок 13 – Полученная маска

Рисунок 14 – Результат работы алгоритма K-Means

Для измерения времени выполнения алгоритма использовалась функция «perf_counter()». В процессе эксперимента было выполнено десять замеров времени, результаты которых представлены в таблице 2.

Таблица 2 – Время выполнения алгоритма K-Means

Номер запуска	Время выполнения, с.
1	0.51081
2	0.42334
3	0.44076
4	0.42241
5	0.40821
6	0.40939
7	0.40910
8	0.45448
9	0.42508
10	0.40473

На основе полученных результатов из таблицы 2, среднее время выполнения алгоритма K-Means составило 0,43083 секунд.

По итогам сегментации изображения методом K-Means:

- индекс Жаккара равен 0.478180 или около 47% совпадения сегментированного изображения с размеченной аннотацией;
- коэффициент Дайса равен 0.646984, что соответствует примерно 64% совпадения сегментированного изображения с размеченной аннотацией.

В результате работы алгоритма была выдела существенная часть легких. Происходит это полуавтоматически, а время, затраченное на выполнение алгоритма, сравнительно небольшое, хотя и больше чем при использовании метода пороговой сегментации.

Оператор Собеля

Оператор Собеля был разработан в 1968 году американским ученым Ирвином Собелем. Он используется в обработке изображений для выделения границ, определяя изменения яркости в изображении. Оператор Собеля применяет два фильтра, один для обнаружения границ в горизонтальном направлении, а другой – в вертикальном, что позволяет выявлять контуры объектов на изображении. Для его реализации необходимо выполнить следующие шаги:

Изображение загружается с помощью функции «cv2.imread()». Путь к изображению указывается в виде строки, а флаг «cv2.IMREAD_GRAYSCALE» указывает на необходимость загрузки изображения в градациях серого.

Применяется оператор Собеля для обнаружения границ в горизонтальном направлении, используя «cv2.Sobel()» с параметрами 1, 0, что означает, что вычисляется производная по оси x. Затем оператор Собеля применяется для обнаружения границ в вертикальном направлении с помощью «cv2.Sobel()» с параметрами 0, 1, что указывает на вычисление производной по оси y.

Результаты горизонтального и вертикального операторов Собеля объединяются, используя формулу для вычисления модуля градиента. Это достигается с помощью функций «cv2.sqrt()» и «cv2.addWeighted()», которые позволяют суммировать квадраты значений градиентов.

На этом этапе выполняется пороговая бинаризация, которая, хотя и является необязательной, помогает лучше оценить работу оператора Собеля, и необходима для работы многих функций, требующих бинарного изображения.

Для визуализации результатов используется библиотека «matplotlib», которая позволяет отобразить изображение с обнаруженными границами и бинаризованное изображение (рисунки 15 и 16).

Запускается программа, в результате выполнения которой получен следующий результат:

Рисунок 15 – Результат работы оператора Собеля

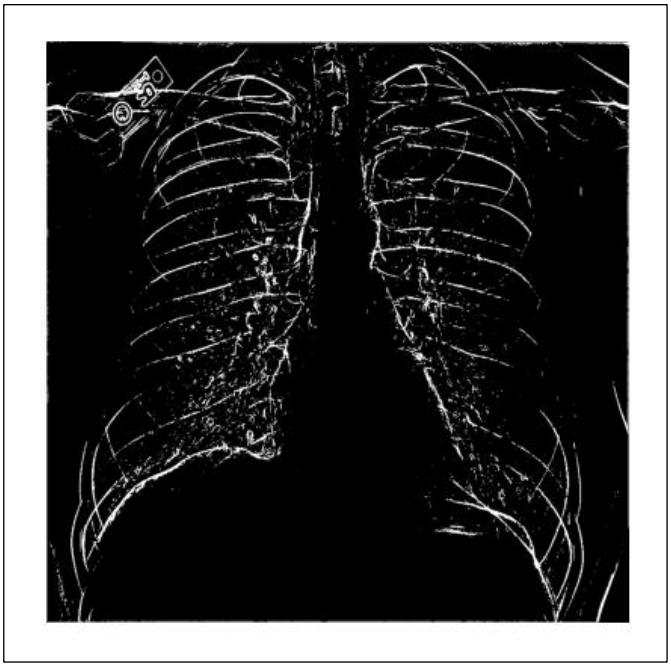

Рисунок 16 – Бинаризованный результат

Измерим время работы алгоритма с применением оператора Собеля. Было выполнено десять замеров времени, результаты которых представлены в таблице 3.

Таблица 3 – Время выполнения алгоритма с применением оператора Собеля

Номер запуска	Время выполнения, с.
1	0.03821
2	0.03971
3	0.04303
4	0.03710
5	0.04161
6	0.04301
7	0.04032
8	0.04060
9	0.03680
10	0.04067

На основе полученных результатов из таблицы 3, среднее время выполнения алгоритма с применением оператора Собеля составило 0,04011 секунд. Этот алгоритм демонстрирует более высокую скорость по сравнению с предыдущими методами сегментации. Однако следует отметить, что оператор Собеля обычно применяется в сочетании с другими методами обработки изображений, что может привести к увеличению общего времени выполнения.

Индекс Жаккара равен 0.063139 или около 6% совпадения сегментированного изображения с размеченной аннотацией. Dice коэффициент равен 0.118778, что соответствует примерно 11% совпадения сегментированного изображения с размеченной аннотацией.

Также метод обнаружения границ с использованием оператора Собеля чувствителен к шумам в изображении. Это связано с тем, что оператор Собеля основан на вычислении градиента яркости, и любые шумы могут привести к значительным изменениям в значениях градиента, что, в свою очередь, может вызвать ложные срабатывания при обнаружении границ.

Поэтому, хоть на получившемся результате области легких визуально хорошо выделены, по результатам вычисления метрик качество сегментации оказалось низким. Из-за большого количества шума, также присутствующего на изображении, использовать данный оператор в работе довольно непросто.

Метод водораздела

Для реализации данного метода сегментации выполняются следующие шаги:

Сначала с помощью функции «io.imread()» загружается снимок, который будет обрабатываться. Второй параметр «as_gray» устанавливается в «True», что необходимо для преобразования цветного изображения в градации серого (рисунок 17). Это упрощает дальнейшую обработку.

Рисунок 17 – Изображение в градациях серого

Найдем область фона, в которой точно нет интересующих нас объектов (легких). При работе метода водораздела данная область будет игнорироваться. Для ее определения воспользуемся методом пороговой сегментации. Значение порога будет установлено на уровне 200, что обусловлено необходимостью избежать случайного захвата интересующих нас объектов.

Для удаления шума применяется функция «remove_small_objects» из модуля «morphology», при этом минимальный размер объектов 5000 пикселей, что означает, что все объекты меньшего размера будут удалены. Получившаяся область выделена белыми пикселями (рисунок 18).

Рисунок 18 – Уверенная область фона

Далее для работы водораздела необходимо задать маркеры. Использование маркеров помогает направить алгоритм, чтобы он знал, где начинать сегментацию, и улучшает качество результата. С помощью функции «np.zeros()», создаем пустое черное изображение (массив нулей), оно будет использоваться для размещения маркеров. В качестве маркеров возьмем небольшие круги, «marker_radius» определяет радиус круга, который будет рисоваться с помощью функции «cv2.circle» из библиотеки «OpenCV», в нашем случае радиус равен 10. Для автоматического присваивания уникальной метки каждому из нарисованных маркеров используется функция «measure.label» из библиотеки «skimage». Незабываем вычесть область фона, в которой мы точно уверены (рисунок 18). Для этого получим максимальную метку с помощью функции «np.max()» и увеличим ее на 1. Теперь присвоим данную метку, со своим цветом, известной нам области фона и наложим эту область на изображение с маркерами.

Для вычисления градиента изображения используем оператор Собеля, функция «filters.sobel» из библиотеки «skimage». После чего применяем метод водораздела, используя функцию «watershed» из модуля «segmentation». С помощью цикла проходим по всем уникальным меткам, полученным после работы водораздела. Для каждой метки создается маска, которая выделяет соответствующую область на исходном изображении. Затем вычисляется средняя яркость пикселей в этой области. Средняя яркость преобразуется в оттенок серого, который затем используется для окраски области в «colored_image». Уникальные значения из «colored_image» сортируются, и создаются интервалы для распределения цветов. Всего задаем 20 интервалов. Для каждого интервала создается маска, и соответствующий цвет из цветовой карты «viridis». После визуализируем сегментированное изображение (рисунок 19).

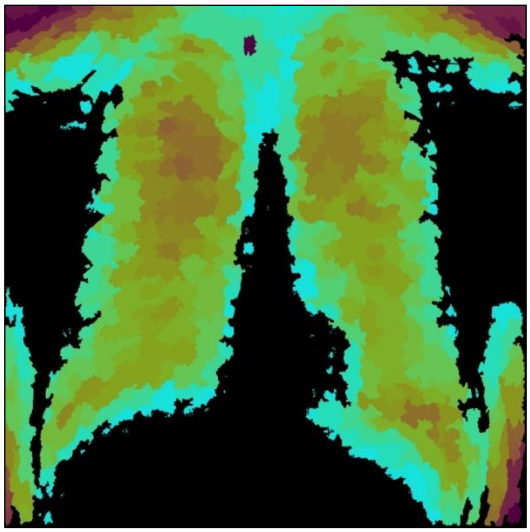

Рисунок 19 – Сегментированное изображение

Далее с помощью функции «addWeighted», накладываем сегментированное изображение (рисунок 19) на оригинальное изображение с заданным уровнем прозрачности, в нашем случае это значение 0,6. Конечный результат работы алгоритма:

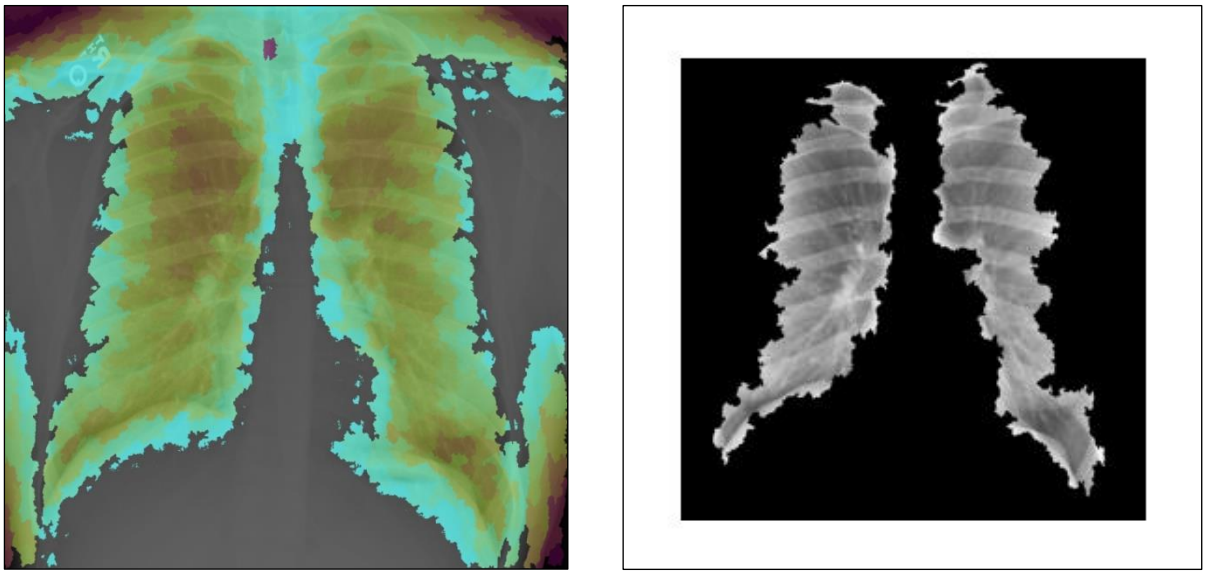

Рисунок 20 – Результат работы метода водораздела

Измерим время работы алгоритма. Было выполнено десять замеров времени, результаты которых представлены в таблице 4.

Таблица 4 – Время выполнения алгоритма водораздела

Номер запуска	Время выполнения, с.
1	4.54166
2	4.49838
3	4.52158
4	4.54695
5	4.63588
6	4.60606
7	4.58328
8	4.53763
9	4.63409
10	4.73107

На основе данных, представленных в таблице 4, среднее время выполнения алгоритма водораздела составило 4,58366 секунды. Следует отметить, что для реализации метода водораздела были использованы сразу несколько дополнительных методов сегментации, включая пороговую сегментацию, оператор Собеля, а также метод нахождения связных областей из библиотеки «skimage». В результате чего, время выполнения алгоритма значительно увеличилось по сравнению с предыдущими подходами к сегментации.

По итогам сегментации изображения методом водораздела:

- индекс Жаккара равен 0.436920 или около 43% совпадения сегментированного изображения с размеченной аннотацией;
- коэффициент Дайса равен 0.608134, что соответствует примерно 60% совпадения сегментированного изображения с размеченной аннотацией.

Как мы можем видеть, метод водораздела выделяет области легких недостаточно эффективно. Но этот метод сложно назвать бесполезным, так например, на сайте pyimagesearch есть полноценный туториал на тему метода водораздела. Как пишет автор, данный алгоритм особенно полезен для небольших соприкасающихся объектов, например для подсчета количества монет на изображении. С практической точки зрения, данный метод применяется в медицине для подсчета кровеносных клеток на снимке.

2.3 Разработка гибридного алгоритма сегментации

Для создания гибридного метода, можно воспользоваться пороговой обработкой, но дополненной методом Отсу для автоматического нахождения порога. Для нахождения связных областей (Connected Components) будем использовать метод «measure.label» из библиотеки «skimage». Свойства областей вычислим с помощью «measure.regionprops». А для улучшения получившихся масок применим морфологические операции.

2.4 Алгоритмы классификации медицинских изображений

В области компьютерного зрения существует множество алгоритмов классификации изображений, которые можно разделить на несколько категорий:

Классические алгоритмы:

- методы опорных векторов (SVM): эффективны для задач с небольшим количеством признаков, но могут быть менее эффективны для сложных изображений, таких как медицинские снимки;
- деревья решений и ансамблевые методы: хорошо работают на структурированных данных, но не всегда подходят для изображений, так как не учитывают пространственные зависимости;
- наивный байесовский классификатор: простой и быстрый алгоритм, но его эффективность снижается при работе с высокоразмерными данными, такими как изображения.

Глубокие нейронные сети:

- сверточные нейронные сети (CNN): это наиболее распространенный подход для классификации изображений. Они автоматически извлекают важные признаки из изображений, что делает их

особенно эффективными для сложных визуальных данных. Самые распространенные сети:

- VGG16: известна своей простотой и высокой точностью, но требует значительных вычислительных ресурсов;
- VGG19: более глубокая версия VGG16, которая может извлекать более сложные признаки;
- ResNet: использует остаточные связи, что позволяет строить очень глубокие сети без проблем с затуханием градиента;
- Inception: модульная архитектура, которая использует различные размеры сверток в одном слое, что позволяет захватывать разные уровни абстракции.

Для задач классификации изображений, особенно в области медицинской диагностики, сверточные нейронные сети (CNN) являются наиболее подходящими. Это связано с несколькими факторами:

- автоматическое извлечение признаков: CNN автоматически извлекают важные признаки из изображений, что позволяет им эффективно обрабатывать сложные визуальные данные, такие как рентгеновские снимки. Классические алгоритмы, как правило, требуют ручного извлечения признаков, что может быть трудоемким и не всегда эффективным;
- глубокая архитектура: глубокие сети, такие как VGG19, ResNet и DenseNet, способны захватывать сложные паттерны и структуры в изображениях благодаря множеству слоев. Это особенно важно для медицинских изображений, где детали могут быть критически важными для точной диагностики;
- доказанная эффективность: сверточные нейронные сети, такие как VGG19, зарекомендовали себя в различных задачах классификации изображений, включая медицинские, что подтверждает их надежность и точность.

Таблица 5 – Сравнение архитектур глубоких нейронных сетей для классификации изображений

Модель	Глубина сети	Преимущества	Недостатки
VGG19	19 слоев	высокая точность; способность извлекать сложные признаки; простота реализации.	высокие вычислительные ресурсы; длительное время обучения.
VGG16	16 слоев	высокая точность; простота реализации.	меньшая глубина, чем VGG19; высокие вычислительные ресурсы; длительное время обучения.
ResNet	50/101 слоев	глубокие сети без затухания градиента; эффективность с меньшим количеством параметров.	сложная архитектура.
Inception v1	22 слоя	модульность; эффективность.	сложность настройки гиперпараметров.

Несмотря на высокие вычислительные требования и длительное время обучения, VGG19 была выбрана как наиболее подходящий алгоритм для нашей задачи, так как она обеспечивает необходимую точность и способность к извлечению сложных признаков, что критично для успешной диагностики на основе рентгеновских снимков.

2.5 Математическая модель сверточной нейронной сети VGG19

VGG19 – это глубокая свёрточная нейронная сеть с 19 слоями, состоящими из 16 свёрточных слоёв, организованных в пять блоков, и 3 полносвязных слоёв. Архитектура представляет собой простую и повторяющуюся структуру, что упрощает её понимание и реализацию.

Ключевыми компонентами архитектуры VGG-19 являются:

1. Свёрточные слои: используются фильтры 3x3 со смещением 1 и отступом 1 для сохранения пространственного разрешения. Каждый свёрточный слой применяет свёртку к входным данным, что можно описать математически по формуле (19).

$$Y_{i,j,k} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X_{(i+m, j+n, c)} \cdot W_{(m, n, k)} + b_k, \quad (19)$$

где X – входное изображение;

Y – выходные данные свёрточного слоя;

W – фильтр (ядро свёртки);

b – смещение;

M и N – размеры фильтра;

i, j – индексы пикселей в выходном изображении;

k – индекс канала.

2. Функция активации: после каждого свёрточного слоя применяется функция активации ReLU, которая вводит нелинейность в модель:

$$f(x) = max(0, x) \quad (20)$$

3. Слой объединения: «MaxPooling» с фильтром 2x2 и шагом 2 для уменьшения пространственных размеров.

4. Полносвязные слои: в конце сети расположены три полностью подключенных слоя, которые выполняют классификацию. Линейная комбинация входов и весов в полносвязном слое вычисляется по формуле (21).

$$s = \sum_{i=1}^{n} w_i \cdot x_i + b, \tag{21}$$

где b – смещение.

После вычисления линейной комбинации применяется функция активации, например, ReLU (22).

$$f(s) = max(0, s) \tag{22}$$

5. Слой Softmax: последний слой для вывода вероятностей классов (23).

$$f(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_i}}, \tag{23}$$

где K – количество классов.

Для бинарной классификации может использоваться сигмоидная функция активации (24).

$$f(x) = \frac{1}{1+e^{-x}} \tag{24}$$

В данной работе сверточная нейронная сеть VGG19 применяется для классификации патологий органов дыхания. Основной задачей обучения является минимизация функции потерь, которая в данном случае является бинарной кросс-энтропией. Математически функцию потерь можно выразить в виде формулы (25).

$$loss = -\frac{1}{N}\sum_{i=1}^{N} y_i \log \hat{y}_i + (1 - y_i) \log (1 - \hat{y}_i) , \quad (25)$$

где N – общее количество изображений в наборе данных;

y_i – истинная метка класса (0 или 1).

Предсказанная вероятность принадлежности к классу 1 вычисляется по формуле (26).

$$\hat{y} = f(\omega \cdot x + b) , \quad (26)$$

где ω – вектор весов;

x – вектор входных признаков;

$f(\omega \cdot x + b)$ – сигмоидная функция активации (24).

3 Реализация и оценка алгоритмов анализа и классификации медицинских изображений

3.1 Реализация гибридного алгоритма сегментации

Для начала необходимо указать путь к изображению и объявить переменную, в которой оно будет храниться. После этого применяется гауссовское размытие для уменьшения шума на исходном снимке. Затем выполняется пороговая обработка, при которой используется метод Отсу для автоматического определения оптимального порога.

После этого вычисляются свойства областей, области сортируются по площади, и выбираются две самые большие из них. На следующем этапе для выбранных площадей необходимо создать маски.

Основная идея заключается в добавлении блока (заполненного белыми пикселями) в выделенную область маски изображения, ограниченную верхним контуром этой области.

Для этого выполним следующие шаги:

1) Инициализация маски: создается начальная маска, которая будет использоваться для дальнейших операций.
2) Нахождение координат пикселей: находятся координаты всех пикселей в «mask_1», которые равны 255.
3) Проверка на пустоту и нахождение границ: если «mask_1» не пустая, то вычисляются максимальная (max_x) и минимальная (min_x) координаты по оси x.
4) Определение координат для добавления блока: переменные «start_x» и «end_x» определяют границы блока, который будет добавлен. «block_width» – это ширина блока, а «block_height» – высота блока, равная высоте маски.
5) Создание блока: создается блок, заполненный единицами (255), который будет добавлен к маске.

6) Нахождение верхнего контура: создается массив «upper_contour_1», который будет хранить верхний контур области. Для каждого x в диапазоне от «start_x» до «end_x» определяется минимальная координата y, где «mask_1» равна 255. Эта координата используется для установки верхнего контура в «upper_contour_1».
7) Добавление блока: в цикле для каждого x снова проверяется, чтобы не выйти за границы. Определяется высота блока, который можно добавить, исходя из верхнего контура. Если «height_to_add» больше 0, то обновляется «mask_1», добавляя блок в соответствующую область.

Процесс обработки второй маски полностью повторяет логику, использованную для первой маски. С полным кодом программы можно ознакомиться в приложении.

После создания масок применяется дилатация для увеличения размеров объектов в маске, что помогает заполнить небольшие пробелы и соединить близкие объекты. В коде это реализуется с помощью функции «cv2.dilate()». Маски «mask_1» и «mask_2» подвергаются дилатации в течение 20 итераций, что позволяет значительно увеличить размеры объектов в масках.

После дилатации применяется морфологическое закрытие, которое помогает устранить небольшие отверстия и улучшить целостность объектов в масках. Это достигается с помощью функции «cv2.morphologyEx()», которая позволяет выполнять различные морфологические операции. В данном случае используется операция закрытия «cv2.MORPH_CLOSE». Маски «mask_1» и «mask_2» преобразуются в тип «uint8» перед применением операции, что является необходимым для корректной работы функции.

После улучшения качества масок они объединяются с помощью операции побитового ИЛИ (cv2.bitwise_or), что позволяет создать комбинированную маску, содержащую все выделенные области. Затем эта объединенная маска применяется к исходному изображению с помощью

операции побитового И (cv2.bitwise_and), что позволяет выделить интересующие области на изображении.

Для проверки корректности работы разработанной программы необходимо провести тестирование полученного алгоритма сегментации.

3.2 Тестирование полученного алгоритма сегментации

В качестве исходного изображения возьмем следующее:

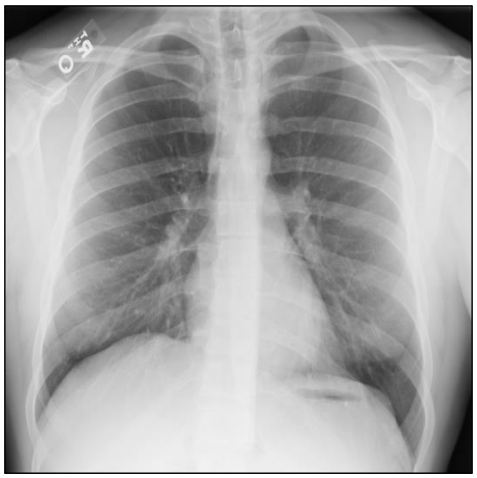

Рисунок 21 – Исходный снимок

Результат работы алгоритма:

Рисунок 22 – Результат работы гибридного метода

Измерим время работы алгоритма гибридного метода. Было выполнено десять замеров времени, результаты которых представлены в таблице 6.

Таблица 6 – Время выполнения алгоритма гибридного метода

Номер запуска	Время выполнения, с.
1	0.39918
2	0.39823
3	0.40866
4	0.42937
5	0.39464
6	0.40678
7	0.40346
8	0.50207
9	0.49159
10	0.40281

На основе данных, представленных в таблице 6, среднее время работы гибридного метода составило 0,42368 секунд. Индекс Жаккара равен 0.835974 или около 83% совпадения сегментированного изображения с размеченной аннотацией. Dice коэффициент равен 0.910660, что соответствует примерно 91% совпадения сегментированного изображения с размеченной аннотацией. Это является хорошими показателями по сравнению с другими методами сегментации.

Таблица 7 – Сравнительные характеристики методов сегментации изображений

Метод	Индекс Жаккара, %	Dice коэффициент, %	Среднее время работы, с.
Пороговая сегментация	48	65	0,07793
K-means	47	64	0,43083
Оператор Собеля	6	11	0,04011
Водораздел	43	60	4,58366
Гибридный метод сегментации	83	91	0,42368

Таким образом, получилось создать алгоритм, состоящий сразу из нескольких методов. Время, затраченное на выполнение алгоритма, оказалось меньше, чем у большинства других методов сегментации. Результаты тестирования алгоритма показали корректность его работы, так как существенная часть легких была выделена. А значит, и поставленная задача была достигнута.

3.3 Обработка DICOM-файлов

В качестве тестируемого снимка будет использоваться снимок грудной клетки в формате DICOM. Снимок считывается из файла с помощью библиотеки «pydicom», используя команду «pydicom.dcmread()», что позволяет загрузить данные изображения в память. При этом данные изображения хранятся в виде массива пикселей, который можно обработать для улучшения визуализации.

Далее происходит нормализация значений пикселей, приводя их в диапазон от 0 до 1. При выводе изображения на экран, оно может выглядеть слишком темным, так как стандартные диапазоны для отображения не всегда подходят для медицинских изображений. Для решения этой проблемы пользователь может ввести значение для коррекции яркости через диалоговое окно, используя функцию «simpledialog.askfloat()». Это значение будет использоваться для настройки контрастности изображения.

Для коррекции яркости и контрастности применяется функция «exposure.rescale_intensity()», которая позволяет задать диапазон значений для преобразования. Переменные «p_min» и «p_max» определяют диапазон значений, который будет использован для контрастирования. После этого применяется адаптивная гистограммная эквализация с помощью функции «exposure.equalize_adapthist()», что позволяет улучшить контраст изображения, выделяя детали, которые могут быть неразличимы на исходном снимке.

После обработки изображение преобразуется в 8-битный формат для отображения. Это позволяет корректно отобразить изображение на экране. Для отображения используется библиотека «PIL», которая позволяет создать объект изображения и отобразить его в графическом интерфейсе.

Если пользователь решает обрезать изображение, он может выделить область на изображении с помощью мыши. Координаты выделенной области сохраняются, и после подтверждения пользователем, изображение обрезается с помощью метода «crop()». После обрезки изображение инвертируется с помощью функции «Image.eval()». Затем обработанное изображение сохраняется в формате PNG с помощью метода «save()».

3.4 Реализация графического интерфейса пользователя

Для удобства работы с алгоритмом сегментации изображений был разработан графический пользовательский интерфейс (GUI). Он позволяет

эффективно обрабатывать медицинские изображения, включая DICOM-файлы, и обеспечивает интуитивно понятный доступ к функционалу сегментации. Интерфейс реализован с использованием библиотеки «tkinter», что позволяет создать удобное и интуитивно понятное приложение для пользователя. Основное окно приложения имеет заголовок «Обработка изображений» и размеры 800x600 пикселей (рисунок 23).

Основная цель интерфейса – упростить процесс сегментации, предоставляя пользователю возможность легко загружать изображения, выбирать области для обработки и визуализировать результаты. Это значительно ускоряет рабочий процесс и снижает вероятность ошибок, связанных с ручным вводом данных.

Рисунок 23 – Основное окно приложения

Элементы интерфейса

а) Кнопки:
   - «Сегментация»: кнопка, расположенная слева, предназначена для открытия диалогового окна выбора изображения, которое будет обработано. При наведении курсора на кнопку цвет фона изменяется на светло-зеленый, что улучшает взаимодействие с пользователем;
   - «Работа с DICOM»: кнопка, расположенная справа, позволяет пользователю выбрать DICOM-файл для обработки. Она также изменяет цвет при наведении;
   - «ОК» и «Отмена»: эти кнопки появляются при выделении области на изображении. Кнопка «ОК» подтверждает выбор области и запускает процесс обрезки изображения, а кнопка «Отмена» скрывает выделение и кнопки.

б) Фреймы:
   - фрейм для кнопок: содержит кнопки и располагается в верхней части окна, заполняя его ширину;
   - фрейм для отображения изображения: заполняет оставшуюся часть окна и предназначен для отображения загруженных и обработанных изображений.

в) Canvas: используется для отображения изображений. Он имеет прозрачный фон и убранные рамки, что позволяет сосредоточиться на изображении. Canvas автоматически подстраивается под размер изображения;

г) Метка для отображения результата: метка, расположенная в фрейме для отображения, используется для показа результатов обработки изображений. Она также очищается перед загрузкой нового изображения.

На рисунке 24 изображено окно выбора файла, при нажатии пользователя на кнопку «Работа с DICOM».

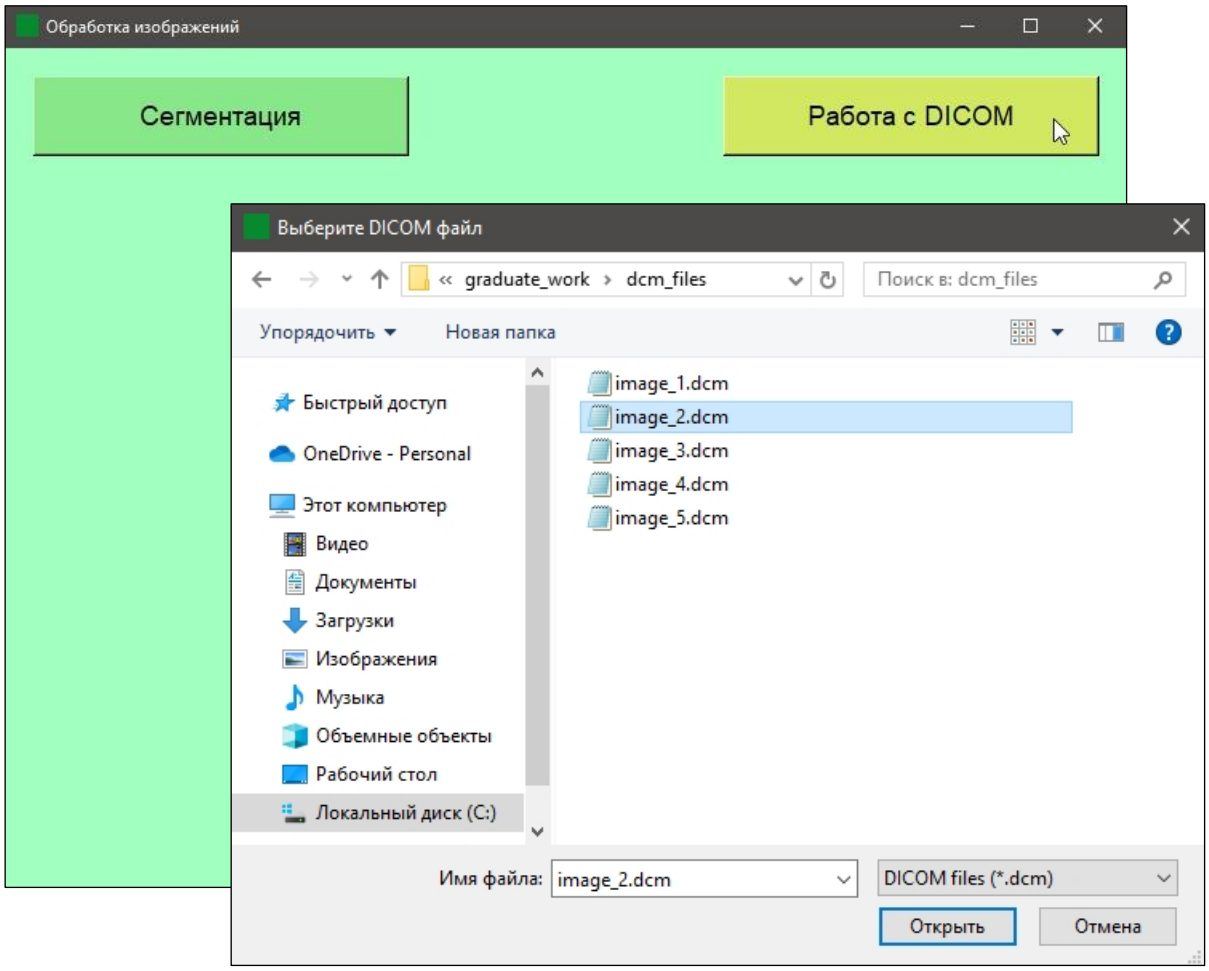

Рисунок 24 – Нажатие на кнопку «Работа с DICOM»

При выделении области появляются кнопки «ОК» и «Отмена», что позволяет пользователю подтвердить или отменить действие (рисунок 25).

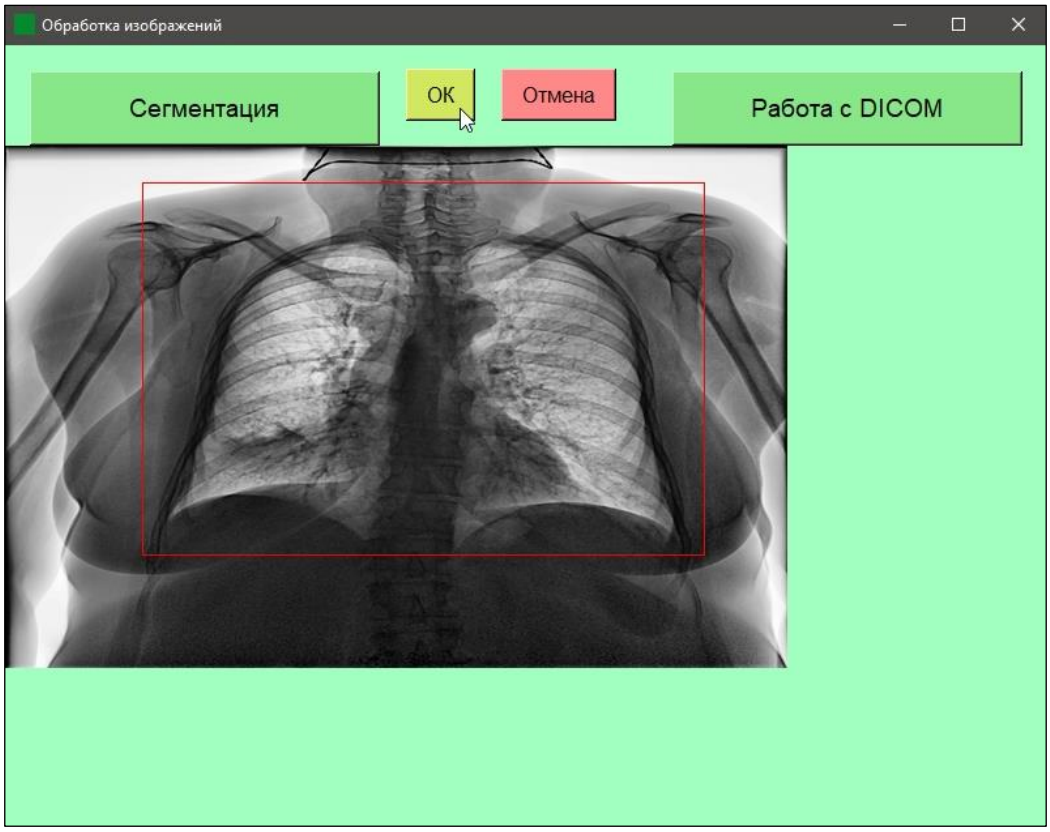

Рисунок 25 – Обрезка изображения

На рисунке 26 показано окно сохранения результата.

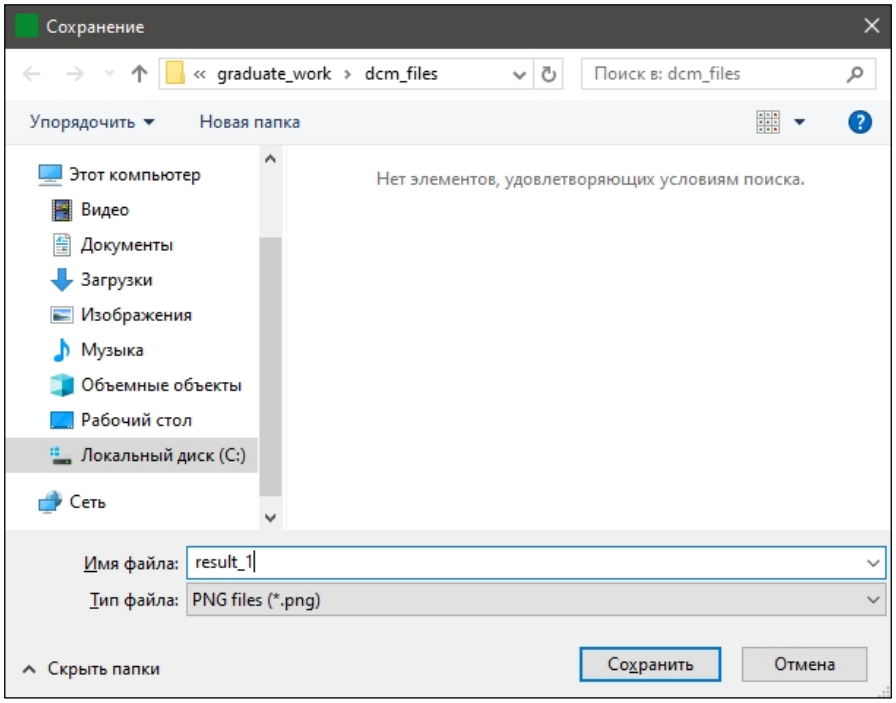

Рисунок 26 – Сохранение результата

На рисунке 27 представлен результат обработки DICOM-файла.

Рисунок 27 – Результат работы (файл result_1.png)

Сегментируем полученный файл (рисунок 28), для этого необходимо нажать на кнопку «Сегментация».

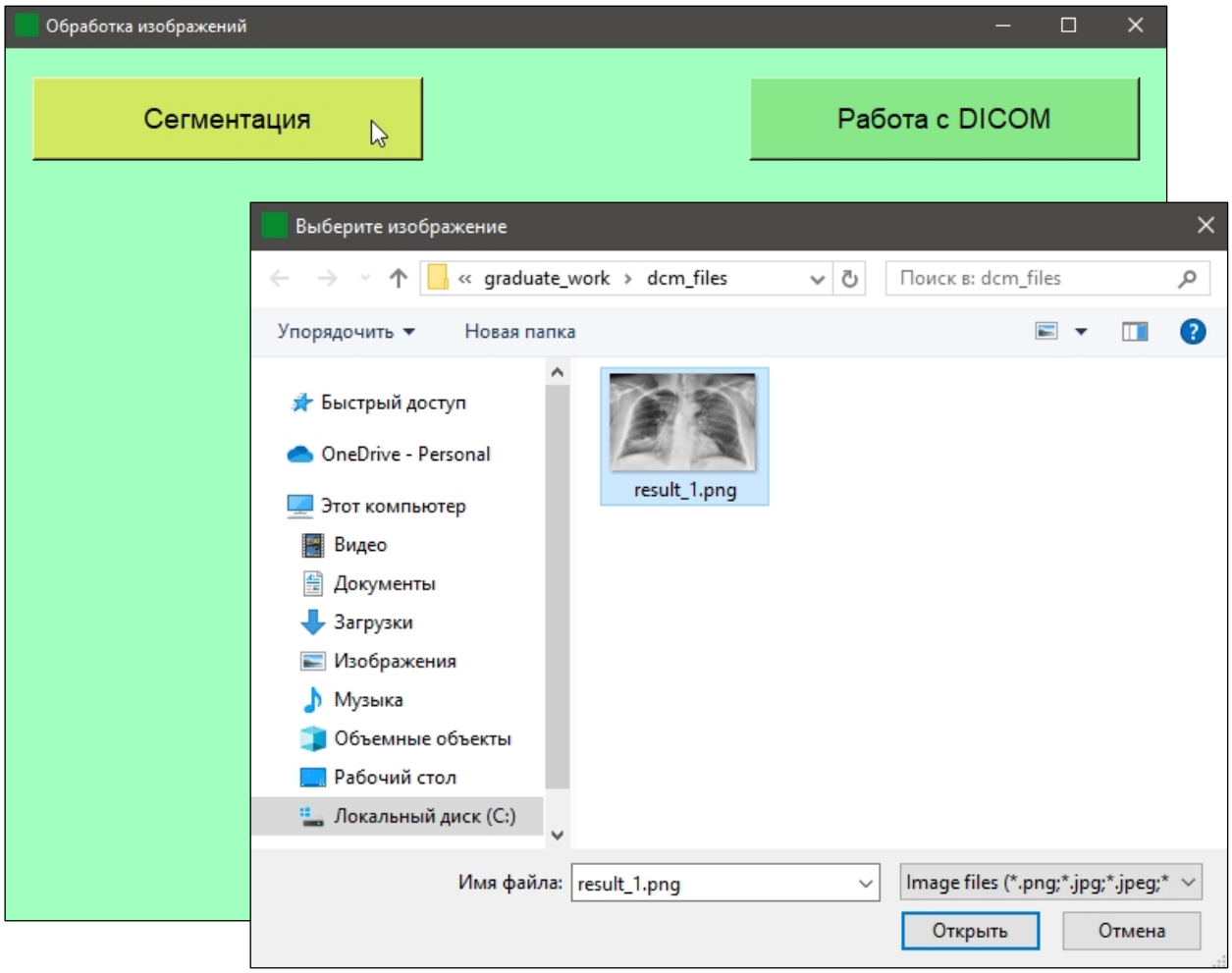

Рисунок 28 – Нажатие на кнопку «Сегментация»

Вывод результата сегментации в окно приложения (рисунок 29).

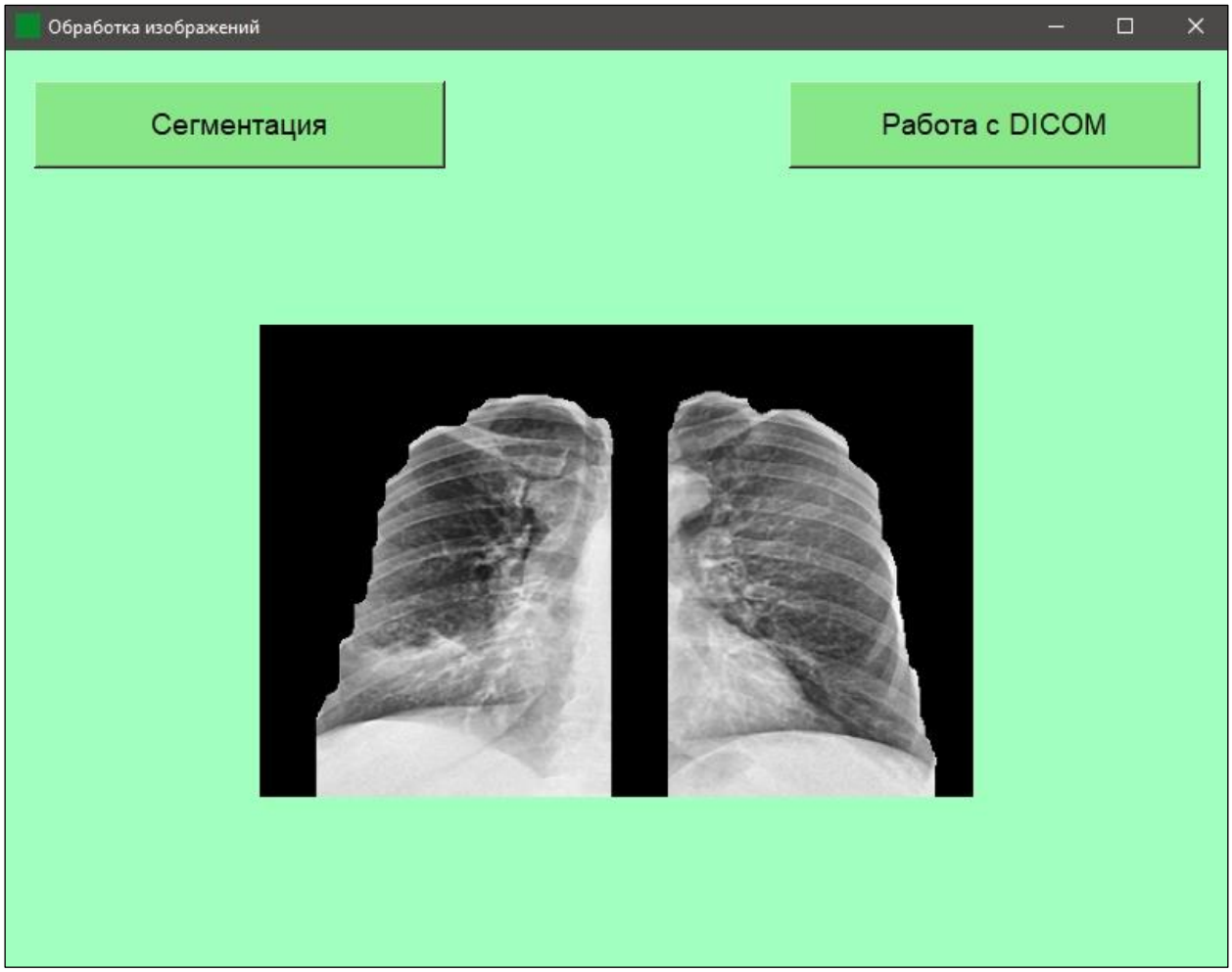

Рисунок 29 – Результат сегментации

3.5 Подготовка данных для нейронной сети

С помощью реализованного гибридного алгоритма сегментации, была проведена сегментация 301 рентгенографического снимка здоровых легких (рисунок 30).

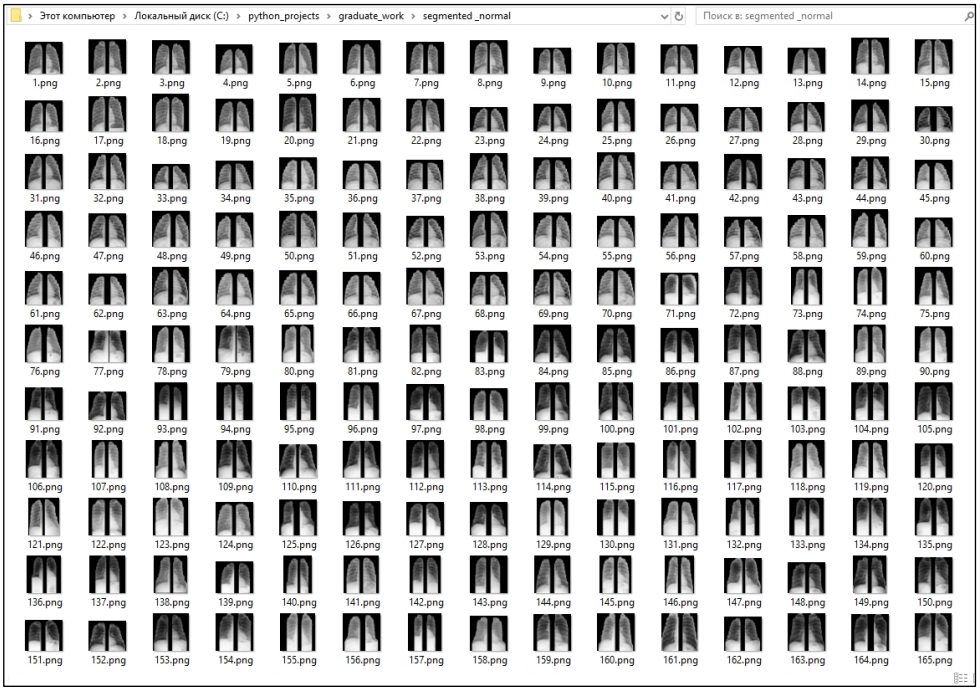

Рисунок 30 – Сегментированные снимки здоровых легких

Также была проведена сегментация 301 рентгенографического снимка легких с пневмонией (рисунок 31).

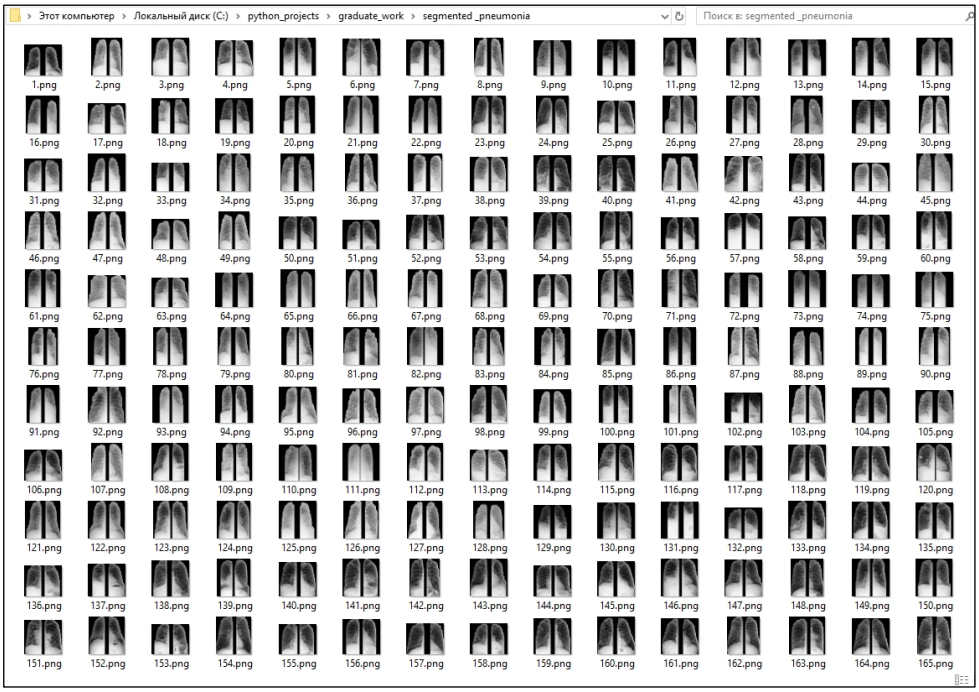

Рисунок 31 – Сегментированные снимки легких с пневмонией

Для увеличения выборки была применена аугментация изображений (рисунок 32). Аугментация изображений – это важный метод в области машинного обучения и компьютерного зрения, который позволяет увеличить объем обучающей выборки. Основные функции аугментации изображений:

- увеличение объема данных: аугментация позволяет создать дополнительные варианты изображений, что помогает избежать переобучения модели на небольшом наборе данных;
- улучшение обобщающей способности: модель, обученная на разнообразных вариантах изображений, будет лучше справляться с новыми, ранее не виденными данными, что повышает ее точность и надежность;
- симуляция различных условий: аугментация может помочь смоделировать различные условия съемки, такие как изменения яркости, повороты и другие трансформации, что делает модель более устойчивой к вариациям в реальных данных.

Примеры аугментации изображений:

- изменение яркости: генерация изображений с измененной яркостью позволяет модели научиться распознавать пневмонию при различных условиях освещения, что может быть полезно в реальных клинических ситуациях;
- повороты: повороты на несколько градусов влево и вправо помогают модели стать более устойчивой к небольшим изменениям в угле съемки, что также может происходить в реальных условиях;
- отражение: отражение изображений не подходит для медицинских снимков, таких как рентгенографии легких. Например, вертикальное отражение может привести к неправильной интерпретации анатомических структур, таких как сердце, которое находится слева. Это может вызвать ошибки в распознавании патологий, так как

модель может неверно идентифицировать увеличенное сердце или другие аномалии.

Важно, чтобы аугментированные изображения сохраняли анатомическую целостность и правильное расположение органов. Неправильные трансформации могут привести к тому, что модель будет обучаться на некорректных данных, что негативно скажется на ее производительности.

```
def augment_image(image):
    augmented_images = []

    augmented_images.append(image)

    for factor in [0.7, 1.3]:
        enhancer = ImageEnhance.Brightness(image)
        bright_image = enhancer.enhance(factor)
        augmented_images.append(bright_image)

    for img in augmented_images[:3]:
        for angle in [-3, 3]:
            rotated_image = img.rotate(angle)
            augmented_images.append(rotated_image)

    return augmented_images
```

Рисунок 32 – Функция аугментации изображений

Функция «augment_image» создает 9 вариантов исходного изображения. Сначала она генерирует два изображения с измененной яркостью: одно с 70% и другое с 130% яркости. Затем каждое из трех изображений (исходное и два с измененной яркостью) поворачивается на 3 градуса влево и вправо. В результате получается набор из 9 изображений, включая оригинал.

Таким образом, всего получаем 301·9=2709 снимков здоровых легких и также 2709 снимков пневмонии легких.

Далее данные необходимо подготовить для обучения сверточной нейронной сети. Основные шаги, которые выполняются в процессе:

Устанавливается основной путь к директории с данными, а также пути к поддиректориям, содержащим нормальные изображения и изображения с пневмонией. С помощью цикла загружаются изображения из каждой категории (нормальные и с пневмонией). Путь к каждому изображению сохраняется в списке «images», а метка (0 или 1) – в списке «labels». Списки «images» и «labels» преобразуются в массивы NumPy для удобства дальнейшей обработки (рисунок 33).

```
# Загрузка изображений и меток
for img_name in os.listdir(normal_dir):
    img_path = os.path.join(normal_dir, img_name)
    img = cv2.imread(img_path)
    images.append(img_path)
    labels.append(0)

for img_name in os.listdir(pneumonia_dir):
    img_path = os.path.join(pneumonia_dir, img_name)
    img = cv2.imread(img_path)
    images.append(img_path)
    labels.append(1)

# Преобразование в массивы NumPy
images = np.array(images)
labels = np.array(labels)
```

Рисунок 33 – Преобразование массивов

С помощью функции «train_test_split» данные разделяются на обучающую выборку (70%) и временную выборку (30%). Параметр «stratify» обеспечивает сохранение пропорций классов в выборках. Временная выборка делится на валидационную (15%) и тестовую (15%) выборки с использованием функции «train_test_split».

Создаются необходимые директории для хранения изображений из обучающей, валидационной и тестовой выборок. Используется «os.makedirs» с параметром «exist_ok=True», чтобы избежать ошибок, если директории уже существуют (рисунок 34). С помощью цикла и «shutil.copy» изображения копируются в соответствующие директории в зависимости от их меток и принадлежности к выборкам (обучающая, валидационная или тестовая).

```
# Создание директорий для обучающей, валидационной и тестовой выборок
os.makedirs(os.path.join(base_dir, 'train/normal'), exist_ok=True)
os.makedirs(os.path.join(base_dir, 'train/pneumonia'), exist_ok=True)
os.makedirs(os.path.join(base_dir, 'val/normal'), exist_ok=True)
os.makedirs(os.path.join(base_dir, 'val/pneumonia'), exist_ok=True)
os.makedirs(os.path.join(base_dir, 'test/normal'), exist_ok=True)
os.makedirs(os.path.join(base_dir, 'test/pneumonia'), exist_ok=True)

# Копирование изображений в соответствующие папки
for img_path, label in zip(images, labels):
    if label == 0:
        if img_path in X_train:
            shutil.copy(img_path, os.path.join(base_dir, 'train/normal'))
        elif img_path in X_val:
            shutil.copy(img_path, os.path.join(base_dir, 'val/normal'))
        else:
            shutil.copy(img_path, os.path.join(base_dir, 'test/normal'))
    else:
        if img_path in X_train:
            shutil.copy(img_path, os.path.join(base_dir, 'train/pneumonia'))
        elif img_path in X_val:
            shutil.copy(img_path, os.path.join(base_dir, 'val/pneumonia'))
        else:
            shutil.copy(img_path, os.path.join(base_dir, 'test/pneumonia'))
```

Рисунок 34 – Создание необходимых директорий

Эти этапы обеспечивают правильную организацию данных для последующего обучения модели на основе сверточных нейронных сетей.

3.6 Реализация сверточной нейронной сети на основе модели VGG19

В данном коде реализуется процесс обучения сверточной нейронной сети VGG19 на собственном наборе данных с использованием библиотек TensorFlow и Keras. TensorFlow предоставляет мощные инструменты для работы с глубоким обучением, а Keras служит высокоуровневым API для упрощения создания и обучения моделей.

Сначала устанавливаются параметры для обработки изображений: высота и ширина изображений равны 224 пикселям, а размер батча составляет 32, что определяет количество изображений, обрабатываемых за один раз. Создаются генераторы данных для обучающей и валидационной выборок с использованием «ImageDataGenerator». Генератор для обучающей выборки включает масштабирование пикселей (нормализация) и заполнение краев изображений, а генератор для валидационной выборки только нормализует изображения. Метод «flow_from_directory» используется для загрузки изображений из указанных директорий, где изображения организованы по подкаталогам, соответствующим классам (например, «нормальные» и «с пневмонией»). Генераторы возвращают батчи изображений и соответствующие метки классов.

Загружается предобученная модель VGG19 без верхнего слоя, что позволяет использовать её как базовую модель для извлечения признаков. Веса модели инициализируются предобученными значениями из набора данных ImageNet. Все слои базовой модели замораживаются, чтобы предотвратить их обновление во время обучения. Это позволяет использовать уже обученные признаки, что особенно полезно при ограниченном объеме данных.

Создается последовательная модель (Sequential), в которую добавляются слои: базовая модель VGG19, слой Flatten, который преобразует выходные данные из многомерного массива в одномерный вектор,

полносвязный слой Dense с 256 нейронами и активацией ReLU для обучения сложных признаков. Слой Dropout с вероятностью 0.5 для регуляризации и предотвращения переобучения, а также выходной слой Dense с одним нейроном и сигмоидной активацией для бинарной классификации (рисунок 35).

```
model = Sequential()
model.add(base_model)
model.add(Flatten())
model.add(Dense( units: 256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense( units: 1, activation='sigmoid'))
```

Рисунок 35 – Слои модели

Модель компилируется с использованием оптимизатора Adam и функции потерь «binary_crossentropy», что подходит для задач бинарной классификации. В качестве метрики используется точность (accuracy). Модель обучается на обучающей выборке с использованием метода «fit()», где указываются генераторы данных, количество шагов на эпоху, валидационные данные и количество эпох. Обучение продолжается в течение 10 эпох (рисунок 36).

```
# Компиляция модели
model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])

# Обучение модели
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    validation_data=val_generator,
    validation_steps=val_generator.samples // batch_size,
    epochs=10
)
```

Рисунок 36 – Компиляция и обучение модели

3.7 Оптимизация обучения сверточной нейронной сети на основе VGG19 с использованием GPU

При запуске программы, обучение на эпохах занимает значительное время. В общей сложности выполнение программы заняло более 30 минут. Модель VGG19 требует значительных вычислительных ресурсов, что является ожидаемым поведением. По стандарту вычисления будут происходить на CPU. Это может быть медленно, особенно для больших наборов данных. Для повышения скорости вычислений модель VGG19 часто запускается на GPU. Графические процессоры оптимизированы для выполнения параллельных вычислений, что делает их особенно эффективными для обработки больших объемов данных.

Для перевода вычислений на GPU необходимо, чтобы видеокарта поддерживала технологию CUDA. В данном случае установлена видеокарта NVIDIA RTX 3050, которая обладает поддержкой CUDA. Также следует отметить, что последней версией TensorFlow, поддерживающей графический процессор в Windows, является TensorFlow 2.10.

Для корректной работы библиотеки потребуется следующее:

- Python версии 3.7–3.10
- CUDA 11.2.0
- cuDNN 8.1.0 для CUDA 11.0, 11.1 и 11.2

Установлен Python версии 3.10, а также все остальные необходимые компоненты (рисунок 37) для работы с VGG19 на GPU. После установки cuDNN необходимо скопировать содержимое папок bin, include и lib в соответствующие директории CUDA. Затем следует обновить переменные среды, добавив пути к bin, libnvvp и include в переменную Path в системных переменных. Теперь система готова к эффективному обучению модели с использованием графического процессора.

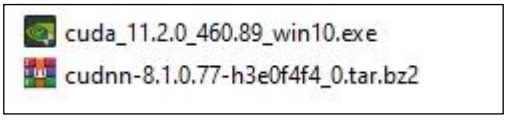

Рисунок 37 – Необходимые компоненты

Теперь время обучение модели занимает не более 5 минут.

3.8 Оценка производительности полученной сверточной нейронной сети на тестовых данных

После успешного обучения модели VGG19, осуществляется оценка её производительности на тестовых данных. Для этого создается генератор тестовых данных с использованием «flow_from_directory», который загружает изображения из указанной директории, устанавливает целевой размер изображений и определяет режим классов (в данном случае бинарный). Параметр «shuffle» установлен в значение False, чтобы сохранить порядок классов (рисунок 38).

```
# Оценка модели на тестовых данных
test_generator = val_datagen.flow_from_directory(
    directory: 'C:/python_projects/dataset_all/test',
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    shuffle=False
)
```

Рисунок 38 – Оценка модели на тестовых данных

Затем извлекаются истинные классы из генератора тестовых данных. Модель оценивается с помощью метода «evaluate», который возвращает значение потерь и точности на тестовом наборе данных. После этого выполняется предсказание классов для тестовых данных с использованием

метода «predict». Полученные предсказания преобразуются в бинарный формат, где значения больше 0.5 интерпретируются как класс 1, а остальные – как класс 0.

Для оценки качества модели рассчитываются метрики: точность (precision), полнота (recall) и F1-мера (F1 score) с использованием соответствующих функций из библиотеки sklearn, формулы которых представлены соответственно в (27), (28) и (29). Результаты выводятся на экран, включая значения этих метрик, что позволяет оценить точность тестирования. Модель и её веса сохраняются в файлы «my_model.h5» и «my_model_weights.h5» соответственно, что позволяет в дальнейшем использовать их без повторного обучения.

$$Precision = \frac{TP}{TP + FP} \tag{27}$$

$$Recall = \frac{TP}{TP + FN} \tag{28}$$

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{29}$$

На рисунке 39 представлены эпохи обучения модели.

```
118/118 [==============================] - 45s 282ms/step - loss: 0.4708 - accuracy: 0.8643 - val_loss: 0.2032 - val_accuracy: 0.9150
Epoch 2/10
118/118 [==============================] - 24s 205ms/step - loss: 0.1768 - accuracy: 0.9313 - val_loss: 0.2020 - val_accuracy: 0.9013
Epoch 3/10
118/118 [==============================] - 24s 201ms/step - loss: 0.1167 - accuracy: 0.9531 - val_loss: 0.1843 - val_accuracy: 0.9125
Epoch 4/10
118/118 [==============================] - 23s 198ms/step - loss: 0.0759 - accuracy: 0.9726 - val_loss: 0.1957 - val_accuracy: 0.9162
Epoch 5/10
118/118 [==============================] - 23s 193ms/step - loss: 0.0702 - accuracy: 0.9731 - val_loss: 0.1826 - val_accuracy: 0.9175
Epoch 6/10
118/118 [==============================] - 23s 195ms/step - loss: 0.0463 - accuracy: 0.9864 - val_loss: 0.1732 - val_accuracy: 0.9200
Epoch 7/10
118/118 [==============================] - 23s 193ms/step - loss: 0.0522 - accuracy: 0.9824 - val_loss: 0.2105 - val_accuracy: 0.9162
Epoch 8/10
118/118 [==============================] - 23s 197ms/step - loss: 0.0380 - accuracy: 0.9883 - val_loss: 0.1883 - val_accuracy: 0.9137
Epoch 9/10
118/118 [==============================] - 23s 195ms/step - loss: 0.0310 - accuracy: 0.9904 - val_loss: 0.1823 - val_accuracy: 0.9212
Epoch 10/10
118/118 [==============================] - 23s 193ms/step - loss: 0.0275 - accuracy: 0.9902 - val_loss: 0.1910 - val_accuracy: 0.9100
```

Рисунок 39 – Эпохи обучения

Точность на тестовой выборке составила около 94% (рисунок 40)

```
Test accuracy: 0.94872
Precision: 0.94161
Recall: 0.95556
F1 Score: 0.94853
```

Рисунок 40 – Метрики

В результате проведенного исследования была успешно обучена модель VGG19 для анализа и классификации патологий на медицинских данных. Оптимизация вычислений с использованием графического процессора позволила значительно сократить время обучения, что повысило эффективность работы. Точность модели на тестовой выборке составила около 94%, что свидетельствует о высокой надежности и качестве разработанного алгоритма. Эти результаты подтверждают целесообразность применения глубокого обучения для анализа медицинских изображений.

Заключение

В ходе выполнения данной работы была разработана система для анализа и классификации патологий на медицинских данных, с акцентом на рентгенографические снимки грудной клетки. Основная проблема заключалась в создании эффективного алгоритма, который бы обеспечивал высокую точность диагностики заболеваний органов дыхания.

Для решения этой задачи была проведена систематизация существующих методов анализа медицинских изображений, включая сегментацию, что является важным этапом для последующей классификации. Были рассмотрены различные подходы, такие как пороговая сегментация, K-Means, оператор Собеля и метод водораздела, что позволило выявить их преимущества и недостатки. На основе проведенного анализа был разработан гибридный метод сегментации, который сочетает в себе пороговую обработку с использованием метода Отсу и морфологические операции. Это позволило значительно улучшить качество сегментации, что, в свою очередь, повысило точность последующей классификации патологий. Тестирование алгоритма сегментации показало высокие результаты: индекс Жаккара составил 83%, а коэффициент Дайса – 91%, что подтверждает эффективность предложенного подхода.

Следующим этапом работы стало создание и обучение сверточной нейронной сети на основе модели VGG19 для классификации изображений. В процессе обучения были рассчитаны ключевые метрики, такие как точность, полнота и F1-мера, что позволило всесторонне оценить качество классификации. Результаты показали, что точность модели на тестовой выборке составила около 94%. Модель и её веса были сохранены в файлы, что позволяет использовать их в дальнейшем без необходимости повторного обучения.

В рамках работы также была реализована система обработки DICOM-файлов и разработан графический интерфейс пользователя, что значительно упростило взаимодействие с системой и сделало процесс анализа более удобным.

Таким образом, результаты данной работы подтверждают, что применение современных алгоритмов анализа и классификации на медицинских данных может значительно улучшить качество диагностики патологий и способствовать более эффективному качеству медицинской помощи. В дальнейшем планируется расширение функционала системы и интеграция с другими методами анализа медицинских изображений для повышения точности и эффективности диагностики патологий.

Список используемых источников

1. Брюс М. Ван Хорн II, Куан Нгуен PyCharm: профессиональная работа на Python / пер. с англ. И. Л. Люско. – Москва: ДМК Пресс, 2024. – 618 с.
2. Вакуленко, С. А. Практический курс по нейронным сетям. / С. А Вакуленко, А. А. Жихарева – Санкт-Петербург: Университет ИТМО, 2018. – 71 с.
3. Вейдман, С. Глубокое обучение: легкая разработка проектов на Python / С. Вейдман. – Санкт-Петербург: Питер, 2021. – 272 с.
4. Гутман, Г.Н. Библиотека Tkinter: графика, геометрия и логические игры на Питоне / Г.Н. Гутман [Электронный ресурс] – URL: https://kpolyakov.spb.ru/download/tkinter_gutman.pdf (дата обращения: 04.06.2025).
5. Дауни, А.Б. Основы Python. Научитесь думать как программист / А.Б. Дауни ; пер. с англ. С. Черникова ; [науч. ред. А. Родионов]. – Москва: Манн, Иванов и Фербер, 2021. – 304 с.
6. Заболеваемость всего населения России в 2023 году: статистические материалы / И.А. Деев, О.С. Кобякова, В.И. Стародубов, Г.А. и др. – Москва: ФГБУ «ЦНИИОИЗ» Минздрава России, 2024. – 154 с.
7. Здравоохранение в России 2023: статистический сборник / под ред. С. М. Окладникова. – Москва: Росстат, 2023. – 179 с.
8. Кошурин, Д.В. Заболевания органов дыхания / Д.В. Кошурин [Электронный ресурс] // Медицинская лаборатория Оптимум : [сайт]. – URL: https://analizy-sochi.ru/spravochnik/zabolevanij-organov-dyhaniya.html (дата обращения: 07.05.2025).
9. Основные алгоритмы сегментации изображений: учебное пособие / Д.Н. Тумаков, З.Д. Каюмов, А.Г. Маркина и др. – Казань: Издательство Казанского университета, 2024. – 76 с.

10. Сверточные нейронные сети / [Электронный ресурс] // proproprogs : [сайт]. – URL: https://proproprogs.ru/neural_network/primery-arhitektur-svertochnyh-setey-vgg16-i-vgg19 (дата обращения: 04.06.2025).

11. Трухан, Д.И. Болезни органов дыхания: актуальные аспекты диагностики и лечения: учебное пособие / Д.И. Трухан, С.Н. Филимонов, Н.В. Багишева. – Новокузнецк, 2020. – 227 с.

12. Тумаков, Д.Н. Технология программирования CUDA: учебное пособие / Д.Н. Тумаков, Д.Е. Чикрин, А.А. Егорчев и др. – Казань: Казанский государственный университет, 2017. – 112 с.

13. Федоров, Д. Ю. Программирование на языке высокого уровня Python: учебное пособие для прикладного бакалавриата / Д. Ю. Федоров. – 2-е изд., перераб. и доп. – Москва: Юрайт, 2019. – 161 с.

14. Time – Time access and conversions / [Электронный ресурс] // Python : [сайт]. – URL: https://docs.python.org/3.10/library/time.html (дата обращения: 03.04.2025).

15. Respiratory Diseases Statistics / [Электронный ресурс] // Eurostat : [сайт]. – URL: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Respiratory_diseases_statistics (дата обращения: 17.05.2025).

16. Image Recognition with Transfer Learning / [Электронный ресурс] // The Data Frog : [сайт]. – URL: https://thedatafrog.com/en/articles/image-recognition-transfer-learning/ (дата обращения: 17.05.2025).

17. Jaccard Index / [Электронный ресурс] // torchmetrics : [сайт]. – URL: https://torchmetrics.readthedocs.io/en/v1.0.0/classification/jaccard_index.html (дата обращения: 25.05.2025).

18. Chest Radiograph Images: Pneumonia and Normal / [Электронный ресурс] // Kaggle : [сайт]. – URL: https://www.kaggle.com/datasets/sangeethakallat/chest-radiograph-images-pneumonia-and-normal (дата обращения: 15.02.2025).

19. Chest X-ray Dataset / [Электронный ресурс] // NIH : [сайт]. – URL: https://nihcc.app.box.com/v/ChestXray-NIHCC (дата обращения: 18.03.2025).

20. Rosebrock, A. Watershed OpenCV / A. Rosebrock [Электронный ресурс] // pyimagesearch : [сайт]. – URL: https://pyimagesearch.com/2015/11/02/ watershed-opencv/ (дата обращения: 11.03.2025).

21. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. 4th ed. – Harlow: Pearson, 2021. 1136 p.

22. Clustering / [Электронный ресурс] // scikit-learn : [сайт]. – URL: https://scikit-learn.org/stable/modules/clustering (дата обращения: 07.04.2025).

23. Euclidean_distances / [Электронный ресурс] // scikit-learn : [сайт]. – URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.

euclidean_distances.html (дата обращения: 19.04.2025).

24. VGG16 and VGG19 / [Электронный ресурс] // Keras : [сайт]. – URL: https://keras.io/api/applications/vgg/ (дата обращения: 07.05.2025).

Приложение А

Листинг программы для обработки и сегментации медицинских изображений

```
import numpy as np
import pydicom
import matplotlib.pyplot as plt
from skimage import exposure
from tkinter import filedialog, simpledialog, messagebox
import cv2
from skimage import measure
import time
import tkinter as tk
from PIL import Image, ImageTk

rect = None
x0, y0 = None, None
image = None
tk_image = None
canvas = None
ok_button = None
cancel_button = None
is_selecting = False

def on_enter(event):
    event.widget.config(bg=«#D2E861»)

def on_leave(event):
    event.widget.config(bg=«#88E788»)

def on_enter_red(event):
    event.widget.config(bg=«#D2E861»)

def on_leave_red(event):
    event.widget.config(bg=«#FF8888»)

def process_image(image_path):
    start_time = time.perf_counter()

    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if image is None:
        raise ValueError(«Не удалось загрузить изображение. Проверьте путь к
файлу.»)

    blurred = cv2.GaussianBlur(image, (5, 5), 0)
    _, thresh = cv2.threshold(blurred, 0, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)
```

```
44
45          labeled_image, num_labels = measure.label(thresh, connectivity=2, return_num=True)
46          regions = measure.regionprops(labeled_image)
47          sorted_regions = sorted(regions, key=lambda r: r.area, reverse=True)
48
49          if len(sorted_regions) < 2:
50              raise ValueError(«Не удалось найти две области на изображении.»)
51
52          largest_regions = sorted_regions[:2]
53          mask1 = np.zeros_like(labeled_image, dtype=np.uint8)
54          mask2 = np.zeros_like(labeled_image, dtype=np.uint8)
55
56          mask1[labeled_image == largest_regions[0].label] = 255
57          mask_1 = mask1
58          y_indices, x_indices = np.where(mask_1 == 255)
59
60          if len(x_indices) > 0:
61              max_x = np.max(x_indices)
62              min_x = np.min(x_indices)
63
64              start_x = min_x
65              end_x = max_x
66              block_width = max_x - min_x
67              block_height = mask_1.shape[0]
68              block = np.ones((block_height, block_width), dtype=np.uint8) * 255
69              upper_contour_1 = np.zeros_like(mask_1, dtype=np.uint8)
70
71              for x in range(start_x, end_x):
72                  if x < mask_1.shape[1]:
73                      upper_contour_y = np.min(np.where(mask_1[:, x] == 255)[0]) if
74      np.any(mask_1[:, x] == 255) else mask_1.shape[0]
75                      upper_contour_1[upper_contour_y:, x] = 255
76
77              for x in range(start_x, end_x):
78                  if x < mask_1.shape[1]:
79                      upper_contour_y = np.min(np.where(upper_contour_1[:, x] == 255)[0]) if
80      np.any(upper_contour_1[:, x] == 255) else mask_1.shape[0]
81                      height_to_add = block_height - upper_contour_y
82
83                      if height_to_add > 0:
84                          mask_1[upper_contour_y:upper_contour_y + height_to_add, x] =
85      np.maximum(
86                              mask_1[upper_contour_y:upper_contour_y + height_to_add, x],
87      block[:height_to_add, x - start_x])
88
89          mask2[labeled_image == largest_regions[1].label] = 255
90          mask_2 = mask2
91          y_indices, x_indices = np.where(mask_2 == 255)
92
93          if len(x_indices) > 0:
94              max_x = np.max(x_indices)
95              min_x = np.min(x_indices)
```

```
            start_x = min_x
            end_x = max_x
            block_width = max_x - min_x
            block_height = mask_2.shape[0]
            block = np.ones((block_height, block_width), dtype=np.uint8) * 255
            upper_contour = np.zeros_like(mask_2, dtype=np.uint8)

            for x in range(start_x, end_x):
                if x < mask_2.shape[1]:
                    upper_contour_y = np.min(np.where(mask_2[:, x] == 255)[0]) if
np.any(mask_2[:, x] == 255) else mask_2.shape[0]
                    upper_contour[upper_contour_y:, x] = 255

            for x in range(start_x, end_x):
                if x < mask_2.shape[1]:
                    upper_contour_y = np.min(np.where(upper_contour[:, x] == 255)[0]) if
np.any(upper_contour[:, x] == 255) else mask_2.shape[0]
                    height_to_add = block_height - upper_contour_y

                    if height_to_add > 0:
                        mask_2[upper_contour_y:upper_contour_y + height_to_add, x] =
np.maximum(
                            mask_2[upper_contour_y:upper_contour_y + height_to_add, x],
block[:height_to_add, x - start_x])

        kernel = np.ones((3, 2), np.uint8)
        mask_1 = cv2.dilate(mask_1, kernel, iterations=20)
        mask_2 = cv2.dilate(mask_2, kernel, iterations=20)
        kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (10, 20))
        mask_1 = cv2.morphologyEx(mask_1.astype(np.uint8), cv2.MORPH_CLOSE, kernel)
        mask_2 = cv2.morphologyEx(mask_2.astype(np.uint8), cv2.MORPH_CLOSE, kernel)

        combined_mask = cv2.bitwise_or(mask_1, mask_2)
        result = cv2.bitwise_and(image, image, mask=combined_mask.astype(np.uint8))
        end_time = time.perf_counter()
        execution_time = end_time - start_time
        print(f»Время выполнения: {execution_time} секунд»)
        return image, thresh, result

    def show_results(result):
        plt.figure(figsize=(6, 6))
        plt.imshow(result, cmap='gray', aspect='equal')
        plt.axis('off')
        plt.savefig('result.png', bbox_inches='tight', pad_inches=0)
        plt.close()
        load_image('result.png')

    def load_image(image_path):
        print(«Загрузка изображения для отображения...»)
        canvas.delete(«all»)
        label_result.config(image='')
        img = Image.open(image_path)
```

```
148         img_tk = ImageTk.PhotoImage(img)
149         canvas.pack_forget()
150         label_result.config(image=img_tk)
151         label_result.image = img_tk
152         label_result.pack(fill=tk.BOTH, expand=True)
153
154     def open_file():
155         file_path = filedialog.askopenfilename(title=«Выберите изображение»,
156     filetypes=[(«Image files», «*.png;*.jpg;*.jpeg;*.bmp»)])
157         if file_path:
158             try:
159                 image, thresh, result = process_image(file_path)
160                 show_results(result)
161             except Exception as e:
162                 print(f»Ошибка: {e}»)
163                 messagebox.showerror(«Ошибка», f»Не удалось обработать изображение:
164     {e}»)
165
166     def cr_DICOM():
167         global image, tk_image, canvas
168         file_path = filedialog.askopenfilename(title='Выберите DICOM файл',
169     filetypes=[('DICOM files', '*.dcm')])
170         ds = pydicom.dcmread(file_path)
171         image_data = ds.pixel_array
172         image_normalized = (image_data - np.min(image_data)) / (np.max(image_data) -
173     np.min(image_data))
174         s = simpledialog.askfloat('Коррекция яркости', 'Введите значение для коррекции
175     яркости (0-255):', minvalue=0, maxvalue=255)
176
177         if s is not None and 0 <= s <= 255:
178             p_min = 0
179             p_max = s / 255
180             adjusted_image = exposure.rescale_intensity(image_normalized, in_range=(p_min,
181     p_max))
182             adjusted_image = exposure.equalize_adapthist(adjusted_image)
183             adjusted_image = (adjusted_image * 255).astype(np.uint8)
184             display_image(adjusted_image)
185
186     def display_image(image_data):
187         global image, tk_image, canvas
188         canvas.delete(«all»)
189         label_result.config(image='')
190         label_result.pack_forget()
191         image = Image.fromarray(image_data)
192         image = image.resize((600, 400), Image.LANCZOS)
193         tk_image = ImageTk.PhotoImage(image)
194         canvas.pack(fill=tk.BOTH, expand=True)
195         canvas.create_image(0, 0, anchor='nw', image=tk_image)
196         canvas.config(scrollregion=canvas.bbox('all'))
197
198     def on_button_press(event):
199         global x0, y0, rect, is_selecting
```

```
200         if not is_selecting and image is not None:
201             x0, y0 = event.x, event.y
202             rect = canvas.create_rectangle(x0, y0, x0, y0, outline='red')
203 
204     def on_mouse_drag(event):
205         global rect, is_selecting
206         if not is_selecting and image is not None:
207             canvas.coords(rect, x0, y0, event.x, event.y)
208 
209     def on_button_release(event):
210         global x0, y0, image, ok_button, cancel_button, is_selecting
211         if not is_selecting and image is not None:
212             x1, y1 = event.x, event.y
213             show_confirmation_buttons(x0, y0, x1, y1)
214 
215     def show_confirmation_buttons(x0, y0, x1, y1):
216         global ok_button, cancel_button, is_selecting
217         is_selecting = True
218         ok_button.pack(side=tk.LEFT, padx=10)
219         cancel_button.pack(side=tk.LEFT, padx=10)
220         ok_button.config(command=lambda: crop_and_save_image(x0, y0, x1, y1))
221         cancel_button.config(command=hide_confirmation_buttons)
222 
223     def hide_confirmation_buttons():
224         global ok_button, cancel_button, rect, is_selecting
225         if rect:
226             canvas.delete(rect)
227             rect = None
228         ok_button.pack_forget()
229         cancel_button.pack_forget()
230         is_selecting = False
231 
232     def crop_and_save_image(x0, y0, x1, y1):
233         global image
234         if image:
235             x0, y0, x1, y1 = map(int, (x0, y0, x1, y1))
236             cropped_image = image.crop((x0, y0, x1, y1))
237             cropped_image = Image.eval(cropped_image, lambda p: 255 - p)
238             save_path = filedialog.asksaveasfilename(defaultextension=«.png»,
239     filetypes=[(«PNG files», «*.png»)])
240             if save_path:
241                 cropped_image.save(save_path)
242             messagebox.showinfo(«Успех», «Изображение успешно сохранено!»)
243             hide_confirmation_buttons()
244 
245     root = tk.Tk()
246     root.title(«Обработка изображений»)
247     root.geometry(«800x600»)
248     root.iconbitmap('iconka.ico')
249     root.config(bg=«#A1FFC0»)
250 
251     button_frame = tk.Frame(root, bg=«#A1FFC0»)
```

```
252         button_frame.pack(fill=tk.X)
253
254         btn_open = tk.Button(button_frame, text=«Сегментация», command=open_file,
255         font=(«Arial», 14), bg=«#88E788», padx=20, pady=10, width=20)
256         btn_open.pack(side=tk.LEFT, padx=(20, 10), pady=(20, 0))
257         btn_open.bind(«<Enter>«, on_enter)
258         btn_open.bind(«<Leave>«, on_leave)
259
260         btn_open2 = tk.Button(button_frame, text=«Работа с DICOM», command=cr_DICOM,
261         font=(«Arial», 14), bg=«#88E788», padx=20, pady=10, width=20)
262         btn_open2.pack(side=tk.RIGHT, padx=(10, 20), pady=(20, 0))
263         btn_open2.bind(«<Enter>«, on_enter)
264         btn_open2.bind(«<Leave>«, on_leave)
265
266         image_frame = tk.Frame(root, bg=«#A1FFC0»)
267         image_frame.pack(fill=tk.BOTH, expand=True)
268
269         canvas = tk.Canvas(image_frame, bg=«#A1FFC0», highlightthickness=0, bd=0)
270         canvas.pack(fill=tk.BOTH, expand=True)
271
272         label_result = tk.Label(image_frame, bg=«#A1FFC0»)
273         label_result.pack(pady=(10, 20))
274
275         ok_button = tk.Button(button_frame, text=«OK», command=None, font=(«Arial», 12),
276         bg=«#88E788», padx=10, pady=5)
277         ok_button.bind(«<Enter>«, on_enter)
278         ok_button.bind(«<Leave>«, on_leave)
279
280         cancel_button = tk.Button(button_frame, text=«Отмена»,
281         command=hide_confirmation_buttons, font=(«Arial», 12),
282                                 bg=«#FF8888», padx=10, pady=5)
283         cancel_button.bind(«<Enter>«, on_enter_red)
284         cancel_button.bind(«<Leave>«, on_leave_red)
285         canvas.bind(«<ButtonPress-1>«, on_button_press)
286         canvas.bind(«<B1-Motion>«, on_mouse_drag)
287         canvas.bind(«<ButtonRelease-1>«, on_button_release)
288        root.mainloop()
```

Приложение Б

Листинг сверточной нейронной сети на основе модели VGG19

```
from sklearn.metrics import precision_score, recall_score, f1_score
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG19
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
import numpy as np

img_height, img_width = 224, 224
batch_size = 32
train_datagen = ImageDataGenerator(rescale=1./255, fill_mode='nearest')
val_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    'C:/python_projects/dataset_all/train',
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary'
)

val_generator = val_datagen.flow_from_directory(
    'C:/python_projects/dataset_all/val',
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary'
)

base_model = VGG19(weights='imagenet', include_top=False, input_shape=(img_height,
img_width, 3))

for layer in base_model.layers:
    layer.trainable = False

model = Sequential()
model.add(base_model)
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])

history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    validation_data=val_generator,
```

```
46          validation_steps=val_generator.samples // batch_size,
47          epochs=10
48      )
49
50      test_generator = val_datagen.flow_from_directory(
51          'C:/python_projects/dataset_all/test',
52          target_size=(img_height, img_width),
53          batch_size=batch_size,
54          class_mode='binary',
55          shuffle=False
56      )
57
58      true_classes = test_generator.classes
59      test_loss, test_accuracy = model.evaluate(test_generator)
60      predictions = model.predict(test_generator, steps=np.ceil(test_generator.samples /
61      batch_size))
62      predicted_classes = (predictions > 0.5).astype(int)
63
64      precision = precision_score(true_classes, predicted_classes)
65      recall = recall_score(true_classes, predicted_classes)
66      f1 = f1_score(true_classes, predicted_classes)
67
68      print(f'Test accuracy: {test_accuracy:.5f}')
69      print(f'Precision: {precision:.5f}')
70      print(f'Recall: {recall:.5f}')
71      print(f'F1 Score: {f1:.5f}')
72
73      model.save('my_model.h5')
74      model.save_weights('my_model_weights.h5')
```