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

РАБОТА ПРОВЕРЕНА

Рецензент
Доцент кафедры ИИТиМОИ
ФГБОУ ВО «ЮУрГГПУ»,
к.п.н., доцент

_____ О.А. Дмитриева
«___»_____ 2024 г.

ДОПУСТИТЬ К ЗАЩИТЕ

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

_____ Л.Б. Соколинский
«___»_____ 2024 г.

Разработка модели промежуточного слияния модальности рентгенологических снимков с мэппингом эмбеддингов из модальности клинических данных для классификации заболеваний легких

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.04.04.2024.308-1487.ВКР

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

Автор работы,
студент группы КЭ-229
_____ Р.Г. Гильманова

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

Челябинск, 2024 г.

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

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

ЗАДАНИЕ
на выполнение выпускной квалификационной работы магистранта

студентке группы КЭ-229
Гильмановой Розалии Галиевне,
обучающейся по направлению
09.04.04 «Программная инженерия»
(магистерская программа «Искусственный интеллект и инженерия данных»)

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)
Разработка модели промежуточного слияния модальности рентгенологических снимков с мэппингом эмбеддингов из модальности клинических данных для классификации заболеваний легких.
2. **Срок сдачи студентом законченной работы:** 20.05.2024 г.
3. **Исходные данные к работе**

3.1. Схема разрабатываемой модели.
3.2. Набор данных в виде файлов рентгенологических снимков.

4. **Перечень подлежащих разработке вопросов**

4.1. Сегментация снимков легких из набора данных для обучения.
4.2. Обучение модели промежуточного слияния модальности рентгенологических снимков на наборе данных из сегментированных и несегментированных рентгенологических снимков.
4.3. Сравнение точности модели при обучении на сегментированных и несегментированных рентгенологических снимках.
4.4. Мэппинг эмбеддингов из модальности клинических данных в модальность рентгенологических снимков.

4.5. Анализ изменения точности модели после мэппинга эмбеддингов модальности клинических данных.

5. Дата выдачи задания: 29.01.2024 г.

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

Задание принял к исполнению Р.Г. Гильманова

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

Ранняя диагностика и прогнозирование развития заболеваний легких имеют решающее значение для эффективного лечения и профилактики заболеваний. При ранней диагностике и прогнозировании пациенты с заболеваниями легких могут получить своевременное лечение, что может привести к лучшим результатам и повышению качества жизни в будущем. Кроме того, раннее вмешательство может помочь предотвратить прогрессирование заболеваний легких, тем самым сократив срок лечения и снизив расходы на здравоохранение.

Нейронные сети могут сыграть важную роль в прогнозировании вероятности развития заболеваний легких. Особенно у лиц, которые потенциально могут подвергаться большему риску из-за таких факторов, как вредные привычки, воздействие плохой экологии или генетической предрасположенности. Нейронные сети могут способствовать выявлению людей с предрасположенностью к развитию заболевания и обеспечить раннее вмешательство для предотвращения или отсрочки появления заболеваний легких.

Основной проблемой при разработке систем обнаружения заболеваний с помощью искусственного интеллекта в медицине является нехватка точных и разнообразных данных. Специфика исходных материалов для обучения заключается в зашумленности и отсутствии покрытия всех возможных типов проявлений болезни [1]. Так, например, шум в КТ-и МРТ-снимки могут вносить как внешние факторы, так и артефакты аппаратуры. Не менее важной особенностью выступает большое количество взаимосвязанных параметров, которые надо учитывать и неопределенность при интерпретации параметров [2].

Также данная область отличается большим и быстро обновляющимся массивом информации. Непрерывно проводятся исследования, на основе которых составляются новые рекомендации и разрабатываются методы

диагностики. Вместе с тем не меньший объем информации устаревает. Более того для анализа зачастую отсутствуют четкие логические взаимосвязи между исходными данными и результатом. В таком случае, использование искусственных нейронных сетей может быть полезным. Применяя методы искусственного интеллекта для анализа больших объемов данных, можно выявить закономерности, которые могут быть упущены человеком-наблюдателем.

Применение ИИ в перспективе способно значительно ускорить постановку диагнозов пациентам, уменьшить количество ошибок при выписывании рецептов. В результате, таким образом, появится возможность снизить нагрузку с медицинского персонала. Следовательно, снизить количество медицинских ошибок, вызванных человеческим фактором. Так как чрезмерная рабочая нагрузка пагубно сказывается на внимании и оперативности принимаемых медиками решений. Использование нейронных сетей для ранней диагностики и прогнозирования заболеваний легких в целом может значительно улучшить качество медицинского обслуживания. В связи с вышеприведенными причинами, для автоматизации и увеличения эффективности диагностики заболеваний в медицине актуально применение методов искусственного интеллекта.

Состояние вопроса исследования

Искусственный интеллект на текущий момент успешно применяется в медицине и здравоохранении уже в течение нескольких десятилетий. Есть исследования доказывающие, что ИИ эффективен при выявлении заболеваний легких, таких как рак легких, хроническая обструктивная болезнь легких (ХОБЛ) и пневмония [3, 4]. Алгоритмы ИИ также могут анализировать образцы дыхания для выявления заболеваний дыхательных путей. Например, исследователи разработали алгоритмы, которые могут анализировать летучие органические соединения (ЛОС) в дыхании человека для выявления рака [5]. Алгоритмы ИИ также способны анализировать голос человека для выявления признаков ХОБЛ и других заболеваний лег-

ких. Обнаружено, что изменения голоса человека, такие как охриплость, могут свидетельствовать о легочных заболеваниях [6].

Предметом исследования являются мультимодальные искусственные нейронные сети. В рамках данной работы используются две модальности – текстовые и графические данные. Эффективность мультимодальных моделей, обучаемых на тексте и графике, уже была доказана при решении других задач [7, 8]. Исследования в данной области направлены на создание мультимодальных моделей, способных анализировать и объединять информацию из различных источников, модальностей, для более точной диагностики, прогнозирования заболеваний, подбора оптимального лечения и т.д. Одним из решаемых в медицине вопросов при исследовании мультимодальных искусственных нейронных сетей является разработка оптимальных методов интеграции данных различных модальностей и создание эффективных архитектур нейронных сетей, способных работать с медицинскими данными.

В целом, основываясь на вышеприведенной информации, исследование мультимодальных искусственных нейронных сетей в медицине представляет большой потенциал для улучшения качества здравоохранения и развития новых методов диагностики и лечения заболеваний, связанных с легкими.

Постановка задачи

Целью выпускной квалификационной работы является разработка модели промежуточного слияния модальности рентгенологических снимков с мэппингом эмбеддингов из модальности клинических данных для классификации заболеваний легких. Для достижения поставленной цели необходимо решить следующие задачи:

1) сегментировать снимки легких из набора данных для обучения;

2) разработать модель промежуточного слияния модальности сегментированных и несегментированных рентгенологических снимков с модальностью клинических данных;

3) сравнить точность модели при обучении на сегментированных и несегментированных рентгенологических снимках;

4) произвести мэппинг эмбеддингов из модальности клинических данных в модальность рентгенологических снимков;

5) проанализировать изменения точности модели после мэппинга эмбеддингов модальности клинических данных.

Структура и содержание работы

Работа состоит из введения, четырех глав, заключения, списка литературы и двух приложений. Объем работы составляет 67 страниц, объем списка литературы – 43 источника.

В первой главе рассматриваются существующие подходы построения мультимодальной искусственной нейронной сети, описывается архитектура сети, разработанной в данной работе.

Вторая глава посвящена описанию проведенной предобработки и анализу ее влияния на точность классификации.

Третья глава описывает архитектуру созданных моделей для классификации данных модальности рентгенологических снимков, клинических данных и классификации на основе обеих перечисленных модальностей.

В четвертой главе проводится анализ качества полученных моделей.

В приложении А содержатся графики метрики accuracy на тренировочной и тестовых выборках и функции потерь, полученные при обучении нескольких моделей с различными архитектурами на одних и тех же двух наборах обучающих данных. Наборы данных при этом составлены из указанного в исходных данных к работе и состоят из предобработанных и непредобработанных семантической сегментацией рентгенологических снимков грудной клетки. В приложении Б содержатся листинги создания моделей.

1. МУЛЬТИМОДАЛЬНЫЕ НЕЙРОННЫЕ СЕТИ

1.1. Описание предметной области

Мультимодальное глубокое обучение – это область искусственного интеллекта, которая включает интеграцию информации из нескольких модальностей, таких как текст, изображения, видео и аудио. Она используется при необходимости обработки данных разного вида. Целью мультимодального глубокого обучения является разработка моделей, которые могут учиться и делать прогнозы на основе данных из разных модальностей. В традиционном глубоком обучении модели обычно работают с одной модальностью, например, с изображением или фрагментом текста. Однако в реальных условиях информация часто представлена в нескольких модальностях. Мультимодальность медицинских данных может проявляться в различных аспектах. Во-первых, когда содержательно идентичная информация фиксируется в разных доменах. Например, медицинский диагноз может потребовать анализа, как медицинских изображений, так и клинических записей. Во-вторых, когда данные внутри одного домена имеют разные статистические характеристики. Например, КТ-изображения, полученные с разнотипного оборудования [9].

Мультимодальное глубокое обучение предлагает несколько преимуществ по сравнению с традиционными подходами к глубокому обучению. Во-первых, он позволяет проводить более полный анализ сложных наборов данных за счет включения информации из нескольких источников. Кроме того, он может повысить точность и надежность моделей за счет использования дополнительной информации из нескольких модальностей или же улучшить производительность [7, 10].

Одним из распространенных подходов к мультимодальному глубокому обучению является использование нейронных сетей, специально разработанных для решаемой задачи. Другой вариант заключается в использовании трансферного обучения, при котором предварительно обученные на отдельных модальностях модели объединяются для формирования

мультимодальной модели. При переносе обученных слоев их веса замораживаются, после чего полученные не тренируемые слои добавляются в результирующую модель [11, с. 197]. Трансферное обучение может быть особенно эффективным, когда для каждой модальности доступны ограниченные данные.

Слияние модальностей

Объединение данных из разных модальностей в архитектурах мультимодальных искусственных нейронных сетей может происходить на различных этапах обучения. Выделяют три варианта объединений:

1) early fusion (раннее слияние);
2) intermediate fusion (промежуточное слияние);
3) late fusion (позднее слияние).

На рисунке 1 приведены схемы вариантов слияния.

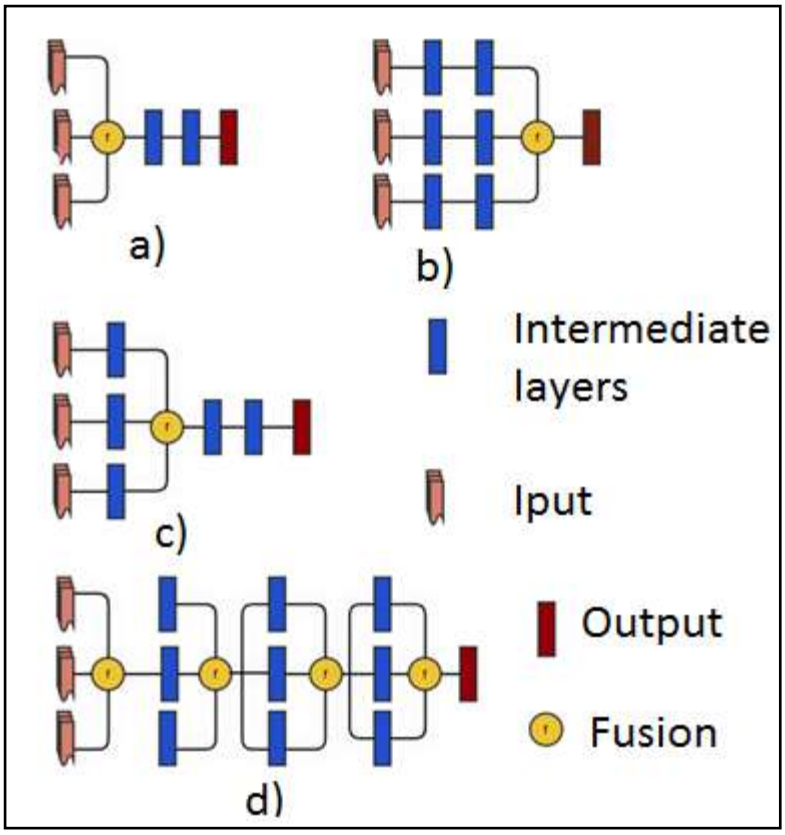

Рисунок 1 – Схемы слияния [12]

Раннее слияние применяется до подачи информации в сеть распознавания. Оно преобразует исходные необработанные данные в более крат-

кую промежуточную форму [9]. Например, визуальные данные могут быть приведены к одному формату и объединены в одно изображение путем наложения одного рисунка на другой. Схема раннего слияния изображена на рисунке 1 под буквой a. Для раннего слияния широко используются методы, базирующиеся на основе математических операций конкатенации и суммирования. Слияние, производимое на основе математической операции суммирования, уменьшает вычислительную нагрузку при обучении, но также при его применении происходит потеря информации. При конкатенации же напротив количество входных данных больше, что увеличивает объем вычислений [12].

Промежуточное слияние – это тип слияния, производимого внутри модели распознавания. Данный вид слияния объединяет особенности, отличающие каждый тип модальности, для создания нового представления, более информативного, чем отдельные модальности, из которых оно возникло. В связи с тем, что данные различных модальностей зачастую дополняют друг друга, объединение признаков, разных видов позволяет одновременно воспользоваться преимуществами нескольких представлений. Что может обеспечить хорошие результаты распознавания по сравнению с использованием модальностей по отдельности [9].

Промежуточное слияние более гибкое и может быть использовано в одном или нескольких слоях [12]. Схемы промежуточного слияния в одном и нескольких слоях изображены на рисунке 1 под буквой c и d соответственно. Промежуточное слияние может быть реализовано различными способами. Например, путем извлечения признаков из разных модальностей, получения на их основе нового совместного представления с помощью конкатенации и полносвязного слоя, встраиванием получившегося представления обратно в части модели, отвечающие за отдельные модальности [13]. Или же возможен вариант с использованием, например, остаточных кросс-модальных связей (cross-modal residual connections). Остаточные кросс-модальные связи – это связи, которые позволяют необрабо-

танному входу одной модальности напрямую взаимодействовать с промежуточными представлениями другой модальности [14]. Хотя среднее слияние охватывает больший объем информации во время обучения, существуют решения с ранним слиянием, которые превосходят промежуточное слияние. Таким образом, оптимальный метод слияния может отличаться в зависимости от задачи [12].

Позднее слияние – архитектура, в которой слияние происходит за пределами моделей мономодальной классификации. Оно объединяет решения каждого классификатора для получения новых, более точных и надежных решений [15]. Схема позднего слияния изображена на рисунке 1 под буквой b. При позднем слиянии данные из нескольких модальностей обрабатываются независимо до этапа объединения, что снижает количество ошибок из-за различий в данных. Таким образом, производительность может увеличиться по сравнению с ранним синтезом. Однако метод позднего слияния может неэффективно использовать всю информацию из разных модальностей, для определенных задач. Поскольку система обрабатывает данные отдельно до момента принятия решения, часть полезной информации может быть не использована [12].

Также возможно применение различных комбинаций раннего, промежуточного и позднего слияний. Иногда его называют комбинированным слиянием. Например, на рисунке 1 под буквой d изображена схема архитектуры с комбинированным слиянием. В нем использованы: метод раннего слияния для одной из модальностей, два промежуточных слияния и позднее слияние.

Из представленных вариантов слияния для реализации был выбран вариант промежуточного объединения (intermediate fusion), как обладающий наиболее разнообразными возможностями при проектировании архитектуры.

1.2. Описание архитектуры разработанной сети

Разработанная искусственная нейронная сеть имеет архитектуру, представленную на рисунке 2.

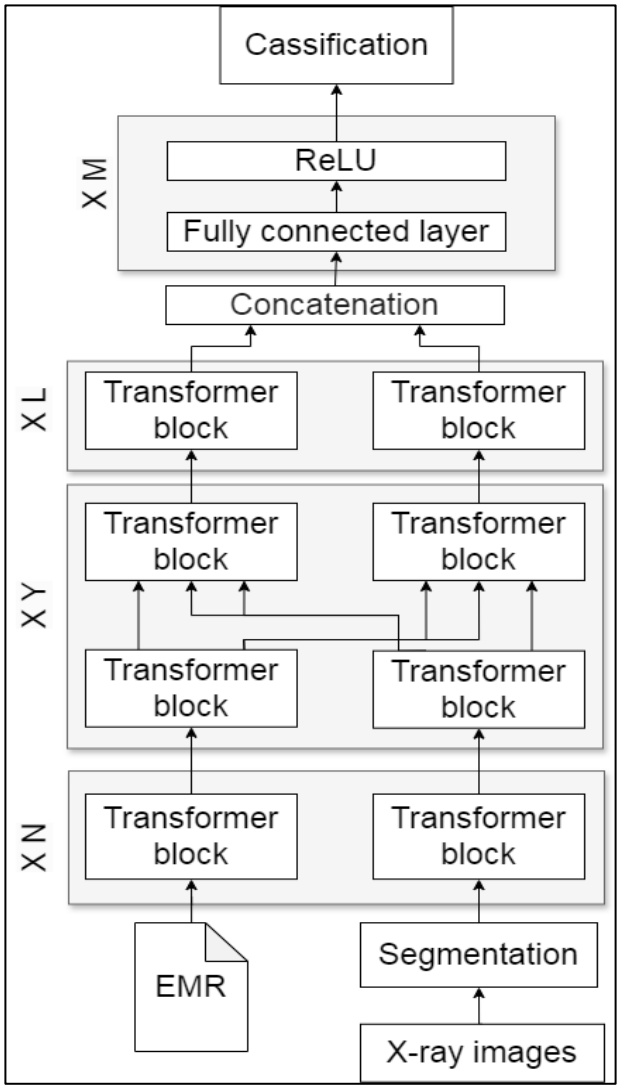

Рисунок 2 – Архитектура разрабатываемой сети

Как показано на схеме, сеть имеет мультимодальную архитектуру для обучения, в которой используются данные двух модальностей. Первая модальность заключается в текстовых файлах с результатами анализов пациентов. Вторая модальность – изображения рентгеновских снимков легких.

В целом, архитектуру сети можно разбить на три составляющие:

1) часть классификации текстовых отчетов о результатах медицинских лабораторных исследований;

2) часть классификации изображений рентгеновских снимков легких;

3) часть объединения данных обучения по различным модальностям.

Как указано на рисунке 2 в реализации сети используются блоки из модели, имеющей архитектуру трансформер.

Сети-трансформеры – это относительно недавняя разработка в области глубокого обучения, которая особенно полезна для задач обработки естественного языка. Основным элементом сетей трансформеров является механизм внимания (attention mechanism), который позволяет модели фокусироваться на различных частях входных данных в зависимости от их значимости для решаемой задачи. Это позволяет сетям трансформерам эффективно работать с последовательными данными, такими как тексты, временные ряды и другие. В медицине сети-трансформеры могут использоваться для анализа медицинских текстовых данных, таких как электронные медицинские карты или медицинские журналы.

Блоки, обозначенные буквами XN, XL, XY и XM, могут повторяться неограниченное число раз, необходимое для каждого конкретного исследования.

В блоках XY происходит перекрестный обмен данными эмбеддингов в трансформерах, обрабатывающих разные модальности.

Вывод по главе 1

На основе приведенной информации можно сделать вывод о том, что использование мультимодальной архитектуры для классификации заболеваний легких, может дать результаты лучше, чем при обучении отдельно на данных одной модальности. При этом использование промежуточного слияния даст больше возможностей для подбора оптимальной архитектуры, а также избавит от потери информации, свойственной раннему и позднему слияниям.

2. ПРЕДОБРАБОТКА РЕНТГЕНОЛОГИЧЕСКИХ СНИМКОВ

Для классификации изображений, возможно, использовать нейронные сети с различными архитектурами. Изначально был распространен метод SIFT + FV. Он включает в себя извлечение признаков SIFT из изображения, их кодирование с помощью FV и использование классификатора для прогнозирования класса изображения [16]. Но на данный момент наиболее широко используются сверточные нейронные сети (CNN) и глубокие сверточные нейронные сети (DCN). Также ведутся разработки по использованию архитектуры трансформеров для работы с изображениями Vision Transformer (ViT).

2.1. Предобработка обучающих данных

Для улучшения результатов классификации была проведена предобработка обучающих данных, состоящих из рентгенологических снимков. Было выдвинуто предположение, что отсечение неинформативных областей изображения, в данном случае, областей, на которых нет непосредственно легких, положительно повлияет на качество обучения сети, так как в этом случае не будет шума в виде неинформативных данных. Для отсечения областей, не содержащих легкие, необходимо решить задачу нахождения объектов, в данном случае легких на снимках.

Задачу нахождения объектов на изображении можно разделить на семантическую сегментацию (semantic segmentation), детекцию объектов (object detection) и сегментацию экземпляров (instance segmentation). При детекции объектов определяются координаты расположения объекта одного из искомых классов. На результирующем изображении найденный объект выделяется рамкой. Такую рамку принято называть bounding box [17]. При семантической сегментации (семантической классификации) определяется принадлежность каждого пикселя к определенному классу. При этом каждый объект искомого класса в отдельности не выделяется. В этом методе, в отличие от детекции объектов, определяется форма границ объ-

ектов. Сегментация экземпляров – самая сложная из сегментаций. При данной сегментации определяется не только принадлежность каждого пикселя к определенному классу, но и выделение отдельно каждого экземпляра класса, то есть отделение объектов одного класса друг от друга [18]. Для предобработки использована семантическая сегментация в связи с тем, что специфика данных исключает возможность присутствия более одного сегментируемого экземпляра на изображении.

2.2. Сегментация легких

На данный момент для работы с изображениями в медицине наиболее популярна архитектура U-Net, т.к. она обеспечивает большую точность в мелких деталях, это наиболее подходящая архитектура для решаемой задачи.

U-Net – это улучшение обычной FCN. Поскольку в процессе свертки информация теряется, для увеличения точности в U-Net присутствует связь блока свертки и соответствующего ему блока повышения дискретизации. Такие связи принято называть shortcut связями. Схематичное изображение U-Net архитектуры приведено на рисунке 3.

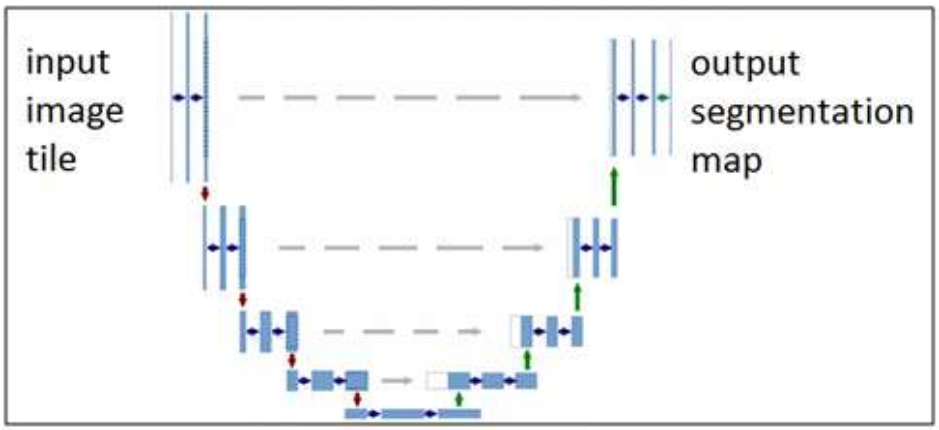

Рисунок 3 – Схема архитектуры U-Net [14]

Shortcut связь восполняет потери и помогает восстановить контуры изображения с большим качеством. Она реализуется путем конкатенации матриц на одинаковых уровнях повышения и понижения дискретизации.

На схеме архитектуры shortcut связи обозначены пунктирными стрелками. После конкатенации производится две последовательные свертки, чтобы модель научилась выдавать более точные результаты. Преимущества данной архитектуры: для обучения достаточно небольшого объема данных, точность в мелких деталях [19].

Для проверки влияния различных методов предобработки на конечный результат, было проведено тестовое обучение предобученных моделей на сегментированных снимках из набора данных, который в последующем используется для классификации [20]. Были протестированы две модели для сегментации. В первом варианте для подготовки данных использована предобученная сеть архитектуры U-Net с сайта kaggle [21]. Данная сеть обучена на датасете «Chest Xray Masks and Labels» [22]. На рисунках 4–6 приведены исходное изображение, маска, полученная при сегментации, и совмещенные маска и исходное изображение соответственно.

Рисунок 4 – Исходное изображение грудной клетки

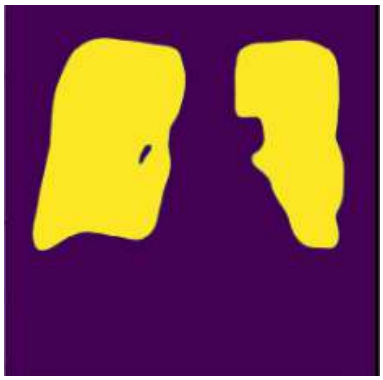

Рисунок 5 – Маска, полученная при сегментации

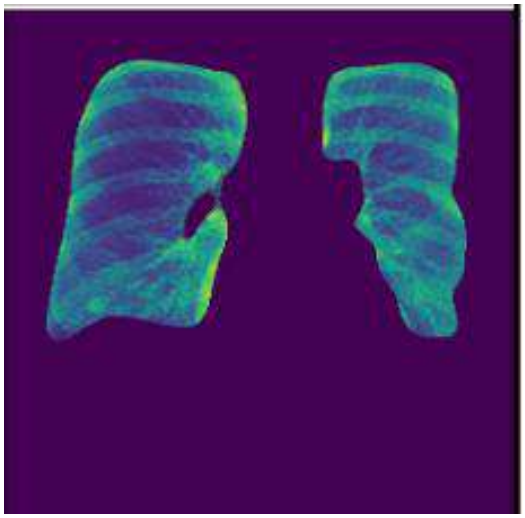

Рисунок 6 – Исходное изображение с наложенной на него маской, полученной при сегментации

В полученных результатах сегментации встречаются явные неточности в виде ошибочного сегментирования близлежащих к легким областей как легких или же напротив не распознавания частей легких. На рисунках 7–8 приведен пример плохо сегментированного снимка, полученной маски и совмещенных маски и исходного изображения соответственно. Предположительно ошибки можно объяснить тем, что сегментируемые снимки имеют меньшее разрешение, чем снимки, на которых обучалась модель. Также могло сказаться плохое качество некоторых исходных снимков.

Рисунок 7 – Исходное изображение плохо отсегментированного снимка

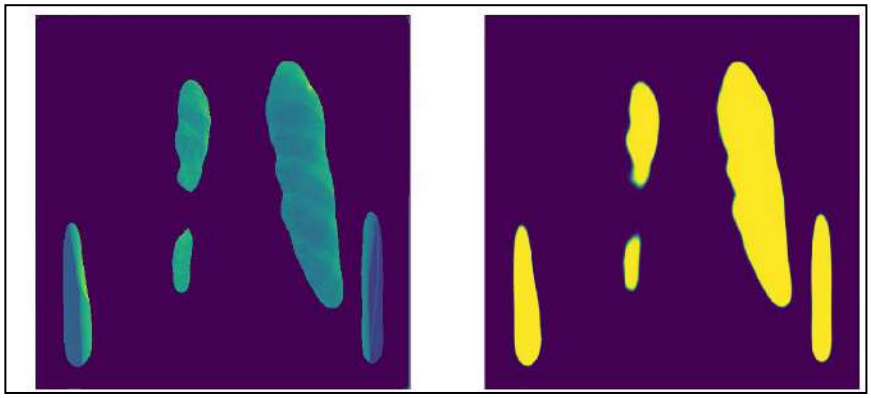

Рисунок 8 – Пример плохого результата сегментации первой модели

Во втором случае была создана модель архитектуры U-Net на основе предобученной VGG16 [23]. В качестве данных для обучения использованы наборы «Lung Masks for Shenzhen Hospital Chest X-ray Set» и «Pulmonary Chest X-Ray Abnormalities» [24, 25]. Создание модели для сегментации и ее обучение произведены с использованием библиотеки keras. Функция создания экземпляра модели приведена на листинге 1 в приложении Б. В функции создания экземпляра в модель vgg16, импортированную из keras.applications, загружаются обученные веса, устанавливается, какие слои должны быть заморожены, а какие – тренироваться в ходе обучения. Также в данной функции создаются shortcut связи.

В ходе обучения использовался оптимизатор Adam. В качестве метрик выбраны binary accuracy и dice coefficient (Коэффициент Серенсена) [26, 27]. В качестве функции потерь используется умноженное на -1 значение коэффициента Серенсена. На листинге 1 приведен код, описывающий функции, вычисляющие dice coefficient и функцию потерь на его основе.

Листинг 1 – Функции, вычисляющие dice coefficient и функцию потерь

```
def dice_coef(y_true, y_pred):
    y_true_f = Flatten()(y_true)
    y_pred_f = Flatten()(y_pred)
    intersection = K.sum(y_true_f * y_pred_f.astype(np.float32))
    return (2. * intersection + 1)/(K.sum(y_true_f) + K.sum(y_pred_f) + 1)

def dice_coef_loss(y_true, y_pred):
    return -dice_coef(y_true, y_pred)
```

В результате обучения модели удалось добиться точности в 67% на валидационной выборке. График изменения binary accuracy при одном из обучений приведен на рисунке 9, график изменения loss приведен на рисунке 10.

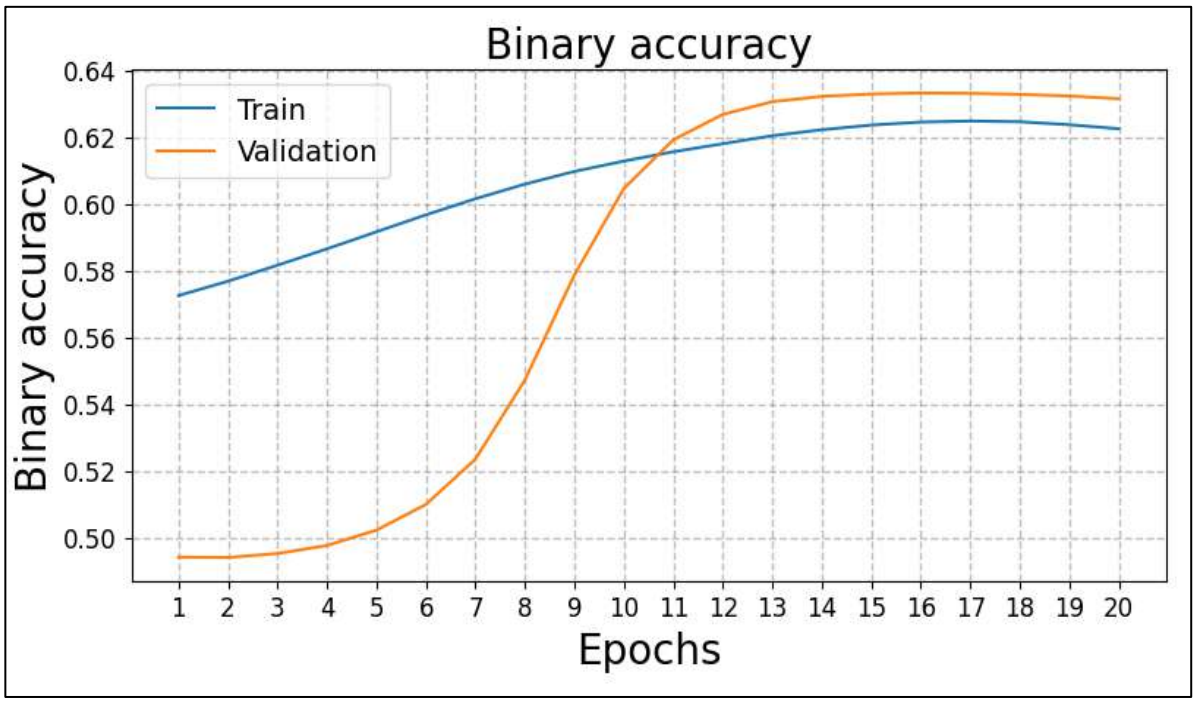

Рисунок 9 – График изменения метрики binary accuracy при сегментации

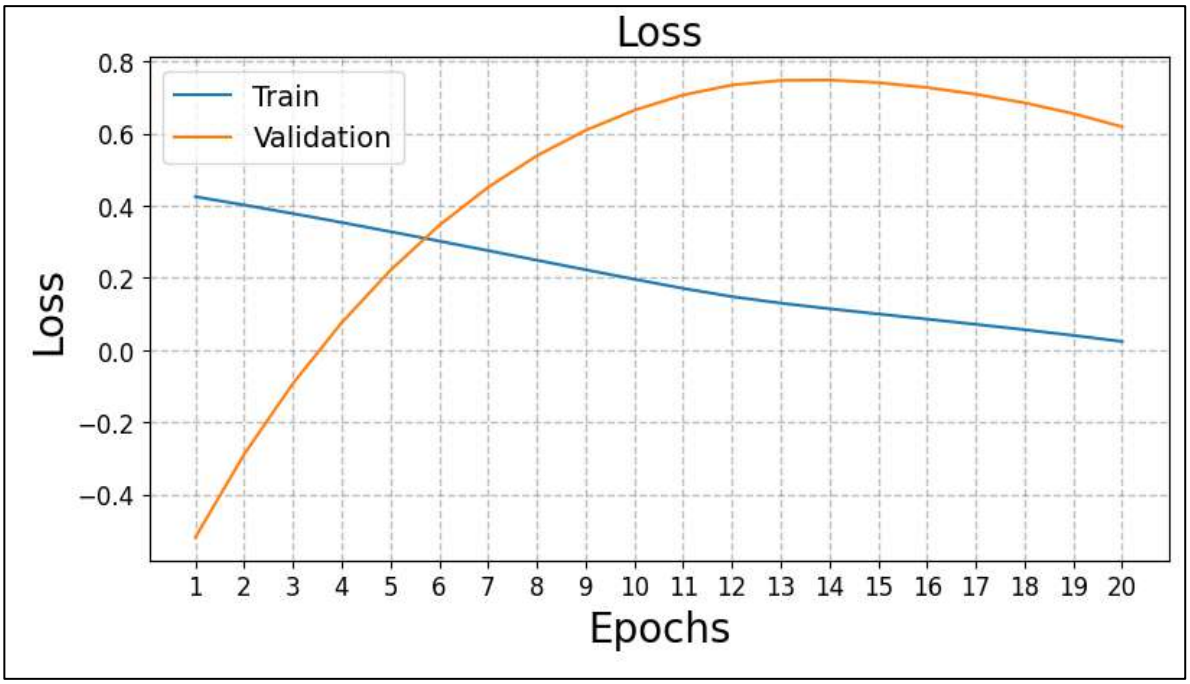

Рисунок 10 – График изменения метрики loss при сегментации

В результатах сегментации набора снимков для классификации моделью на основе VGG16 также встречаются ошибки. На рисунке 11 приведен пример плохо отсегментированного снимка.

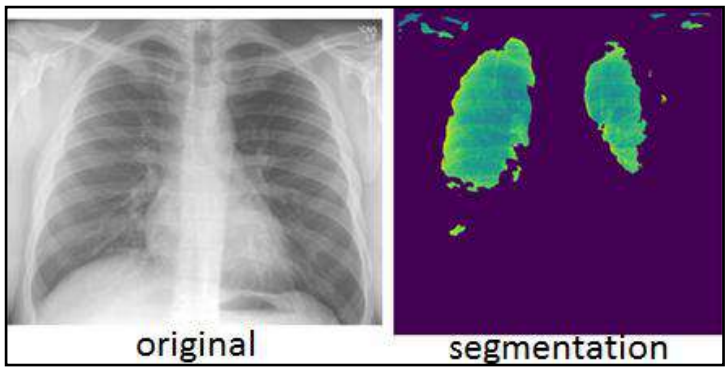

Рисунок 11 – Пример плохого результата сегментации второй модели

Также полученные маски более фрагментированные по сравнению с первым вариантом сегментации. На рисунке 12 приведен пример удачной сегментации легких.

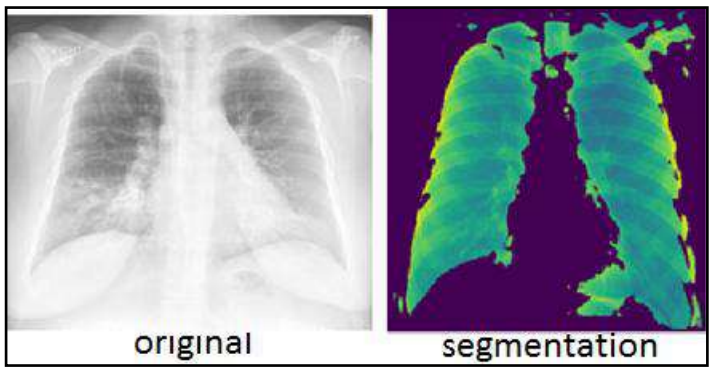

Рисунок 12 – Пример хорошего результата сегментации второй модели

В общем, модель на основе VGG16 склонна скорее к ложноположительным результатам, при которых как легкие определяются области ими не являющиеся. При классификации данный тип ошибки предпочтительней, так как в таком случае менее вероятна потеря важной информации. Принимая во внимание данный факт, а также то, что общее количество ошибок у второй модели было заметно меньше, для сегментации набора данных для классификации было решено использовать вторую модель.

2.3. Анализ влияния сегментации на классификацию

Для теста были взяты предобученные модели из раздела библиотеки keras – Keras Applications [28]. Из доступных моделей были отобраны следующие: DenseNet121, DenseNet169, DenseNet201, VGG19, NASNetLarge.

DenseNet

DenseNet – архитектура, имеющая тип сверточной нейронной сети с прямой связью (CNN), которая связывает каждый уровень с каждым другим слоем. DenseNet состоит из последовательно соединенных переходных уровней и плотных блоков. Каждый сверточный слой внутри плотного блока связан с каждым другим слоем внутри блока. Что достигается путем подключения выхода каждого слоя к входу следующего уровня, создавая «короткую» ссылку. Это позволяет сети более эффективно обучаться за счет повторного использования функций. Пример строения плотного блока приведен на рисунке 13 [29].

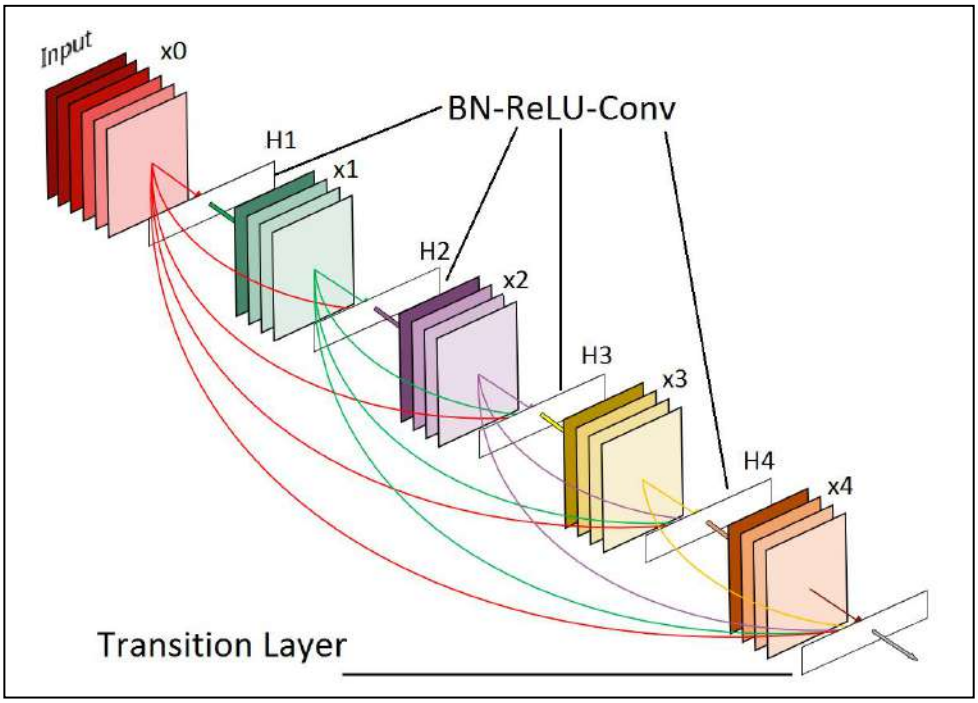

Рисунок 13 – Плотный блок DenseNet с 5 слоями и скоростью роста 4 [29]

DenseNet имеет несколько весомых преимуществ: решает проблему исчезающего градиента, усиливает распространение признаков, поощряет повторное использование признаков и существенно сокращают количество

параметров [30]. А также DenseNet эффективна на малых наборах обучающих данных [31]. DenseNet имеет несколько реализаций: DenseNet121, DenseNet169, DenseNet201. Число в названии модели укзывает на количество, содержащихся в ней плотных блоков.

VGG

VGG (Visual Geometry Group) – архитектура сверточной нейронной сети, улучшенная модель AlexNet. Улучшением является уменьшение количества настраиваемых параметров, достигнутое за счет замены масок сверток равносильными несколькими подряд идущими свертками с меньшим размером масок [32]. На рисунке 14 приведена схема архитектуры сети VGG.

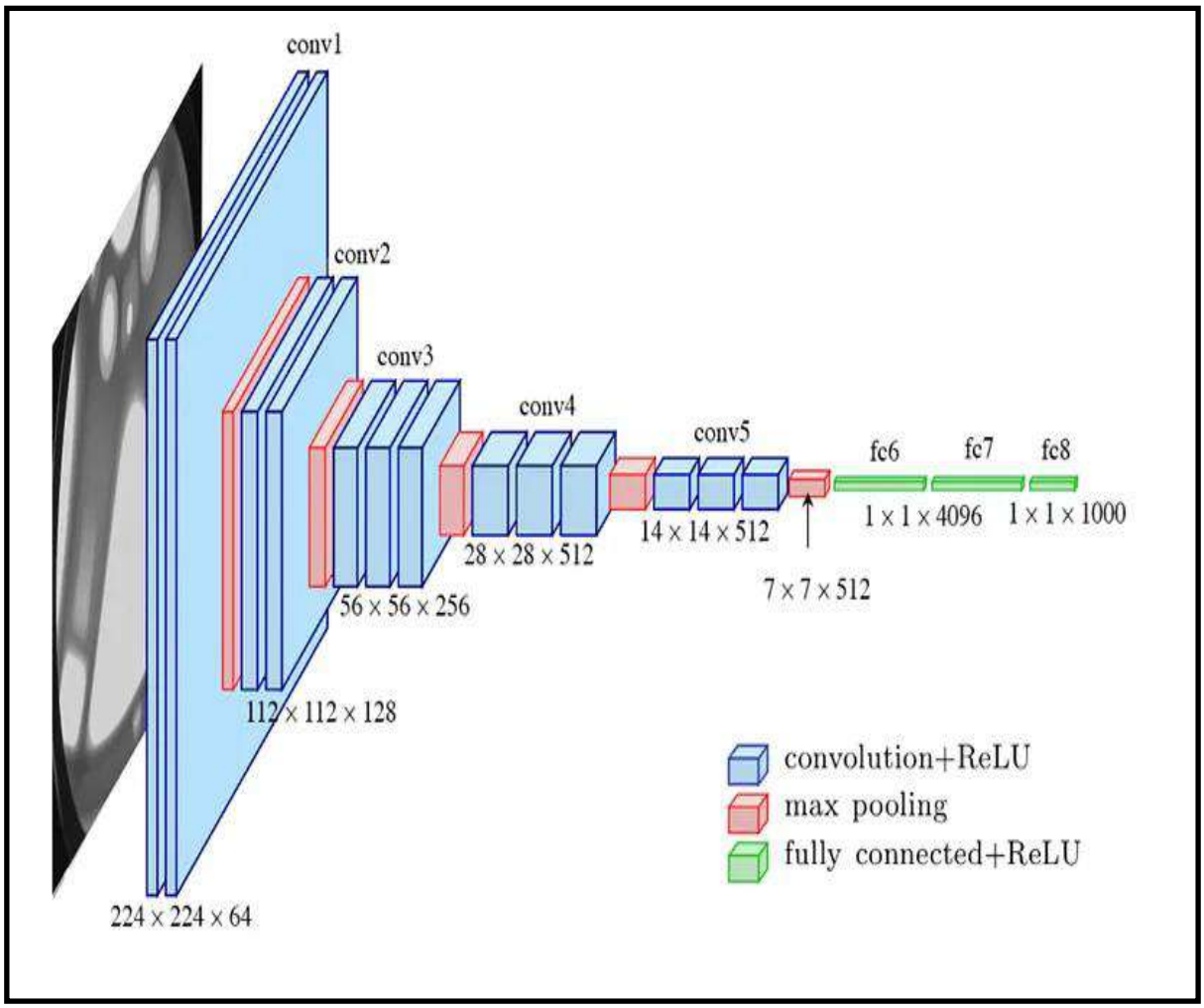

Рисунок 14 – Архитектура сети VGG [33]

VGG модель принимает в качестве входных данных изображения размером 224 на 224 пикселя, которое проходит через convolution слои и функцию активации ReLU. Для линейного преобразования входных данных используются 11 фильтров свертки. Между сверточными слоями производится max pooling. Последние слои состоят из трех слоев с полной связностью и ReLU. Первые два слоя имеют по 4096 каналов каждый. Последний слой имеет по одному каналу на каждый класс, всего 1000 каналов. Существуют две реализации сети – VGG16 и VGG19. Числа в названиях обозначают общее количество слоев в сети [33].

NASNet

NASNet (Neural Architecture Search Network) – это архитектура нейронной сети, разработанная командой Google Brain в 2017 году [34]. Отличительной особенностью NASNet является автоматический поиск лучшей архитектуры, реализованный в виде задачи обучения с подкреплением. При обучении модели происходит поиск лучшей комбинации параметров в пространстве поиска, состоящем, например, из размеров фильтров, выходных каналов, шагов, количества слоев. Общая настройка NASNet включает три компонента: пространство поиска, стратегию поиска, стратегию оценки производительности. Пространство поиска представляет собой пространство всех возможных архитектур, которые следует искать в отношении решаемой задачи. Стратегия оценки производительности заключается в вычислении приближенной производительности нейронной сети на этапе ее проектирования.

Обучение всех моделей производилось в течение пяти эпох. Был использован небольшой набор данных, состоящий из 77 рентгенологических снимков, разделенный на тренировочную и тестовую выборки в соотношении 75% и 25% соответственно. В качестве оптимизатора взят алгоритм adam. В качестве функции потерь выбрана binary_crossentropy. Для оценки эффективности процесса обучения использовались метрики loss и accuracy.

Изменение accuracy для проведенных экспериментов представлено в таблице 1.

Таблица 1 – Значения accuracy

Архитектура	**Обучающие данные**	**Эпоха 1**	**Эпоха 2**	**Эпоха 3**	**Эпоха 4**	**Эпоха 5**	**Изменение accurancy [%]**
DenseNet121	Обработаны	0,6569	0,5882	0,4118	**0,7059**	0,4902	4,4
	Не обработаны	0,5000	0,4706	0,5441	0,3235	**0,6618**	
DenseNet169	Обработаны	0,5000	0,5000	0,5000	0,5000	**0,5000**	0
	Не обработаны	0,2059	0,4559	0,4559	0,4559	**0,5000**	
DenseNet201	Обработаны	0,4020	**0,6471**	0,5000	0,5000	0,5000	13,2
	Не обработаны	0,5000	0,5000	0,5000	**0,5147**	0,5000	
VGG19	Обработаны	0,5000	0,5000	**0,6765**	0,5000	0,5000	17,6
	Не обработаны	0,2647	0,5000	**0,5000**	0,5000	0,5000	
NASNetLarge	Обработаны	0,6961	**0,8431**	0,5686	0,4706	0,7941	18,1
	Не обработаны	0,2059	**0,6618**	0,5000	0,5000	0,5000	

Изменение loss по эпохам обучения для проведенных экспериментов представлено в таблице 2. Графики обучения приведены в приложении А на рисунках 1–10.

Таблица 2 – Значения loss

Архитектура	**Обучающие данные**	**Эпоха 1**	**Эпоха 2**	**Эпоха 3**	**Эпоха 4**	**Эпоха 5**
DenseNet121	Обработаны	1,3437	0,4660	0,4333	**0,4302**	0,5243
	Не обработаны	0,8675	4,5529	2,0352	**0,4319**	0,4645
DenseNet169	Обработаны	1,6512	1,7190	1,4820	**1,3339**	1,6904
	Не обработаны	0,9458	0,5366	**0,3658**	0,4852	1,1177
DenseNet201	Обработаны	0,5511	**0,4374**	0,9899	2,0279	2,0886
	Не обработаны	1,1533	0,5777	3,8164	**0,4467**	12,1581
VGG19	Обработаны	0,5597	0,8669	**0,4230**	0,6946	0,6606
	Не обработаны	0,8491	0,3869	0,9648	**0,3377**	0,8848
NASNetLarge	Обработаны	0,6905	0,5771	**0,5289**	0,6160	0,5881
	Не обработаны	**0,4458**	0,4541	0,8290	1,0096	1,4648

Также для DenseNet121 было проведено обучение на 504 снимках, из которых 378 тренировочных и 126 тестовых. Обучение проводилось в течение 100 эпох. На рисунке 15 приведены графики изменения accuracy, на рисунке 16 – изменения loss с предобработкой в виде сегментации.

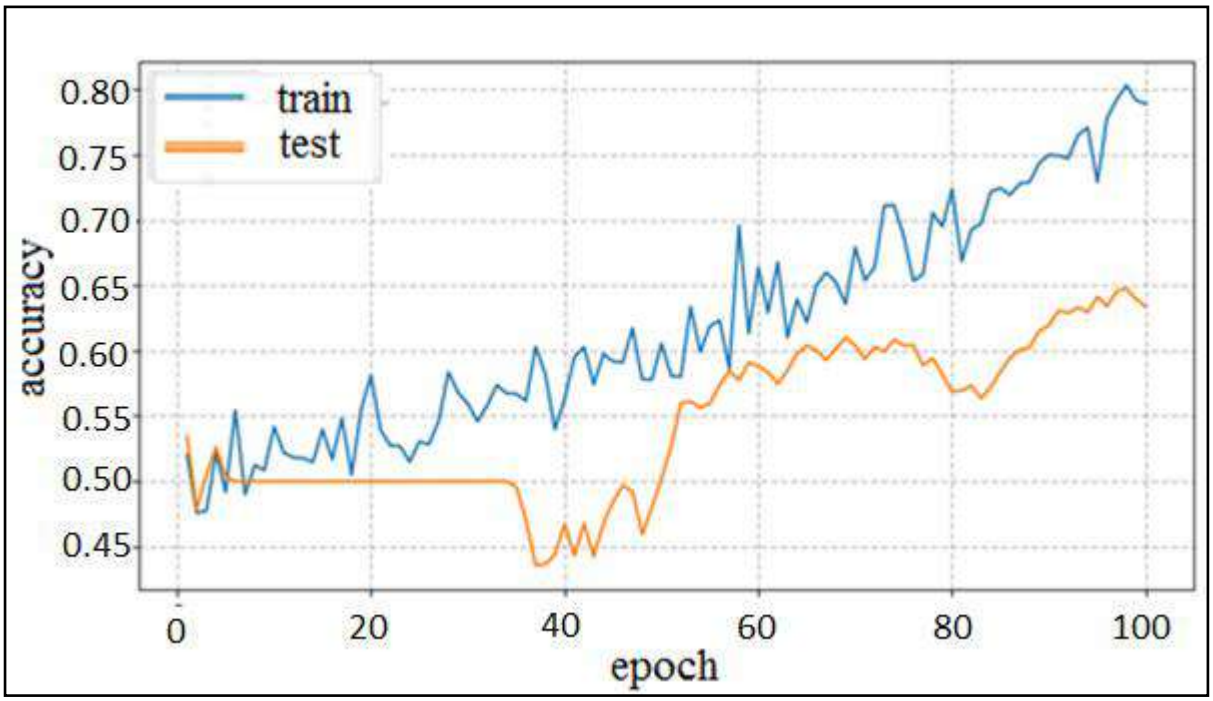

Рисунок 15 – График изменения accuracy при обучении
на несегментированных снимках

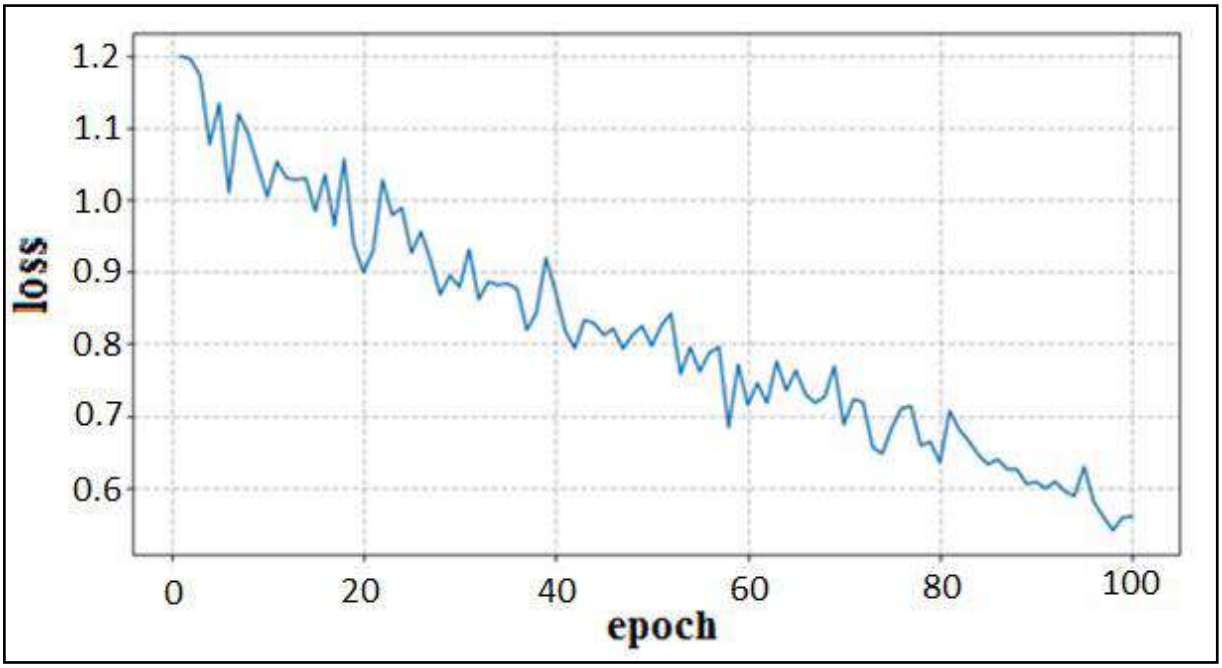

Рисунок 16 – График изменения loss при обучении
на несегментированных снимках

На рисунке 17 приведены графики изменения accuracy, на рисунке 18 – изменения loss без применения предобработки в идее сегментации легких на снимках.

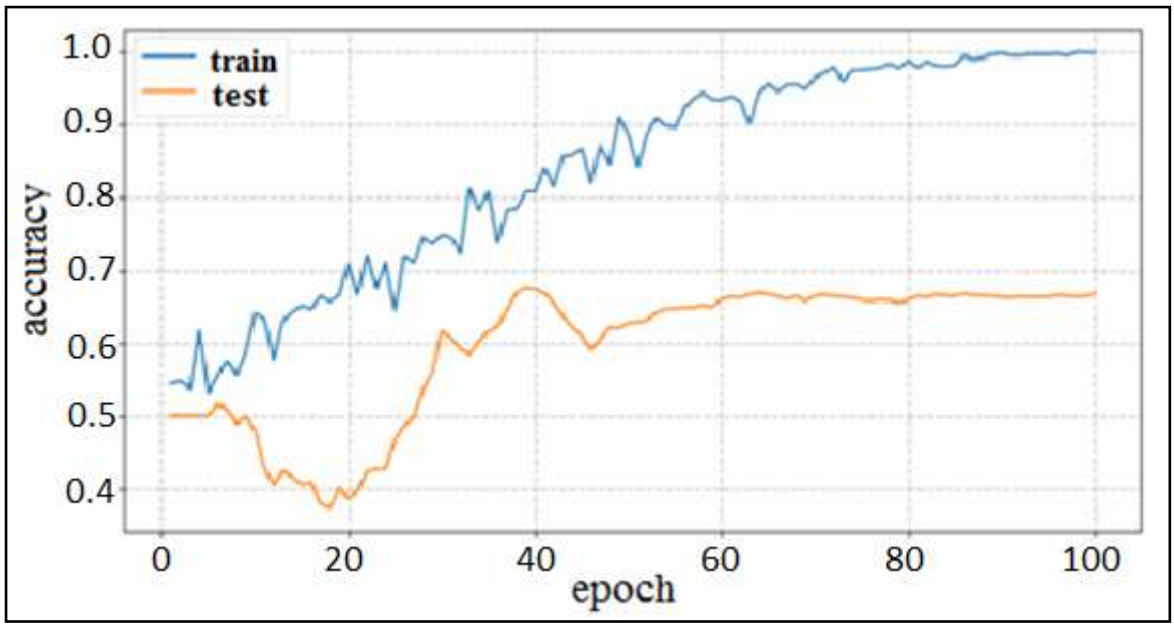

Рисунок 17 – График изменения accuracy при обучении
на сегментированных снимках

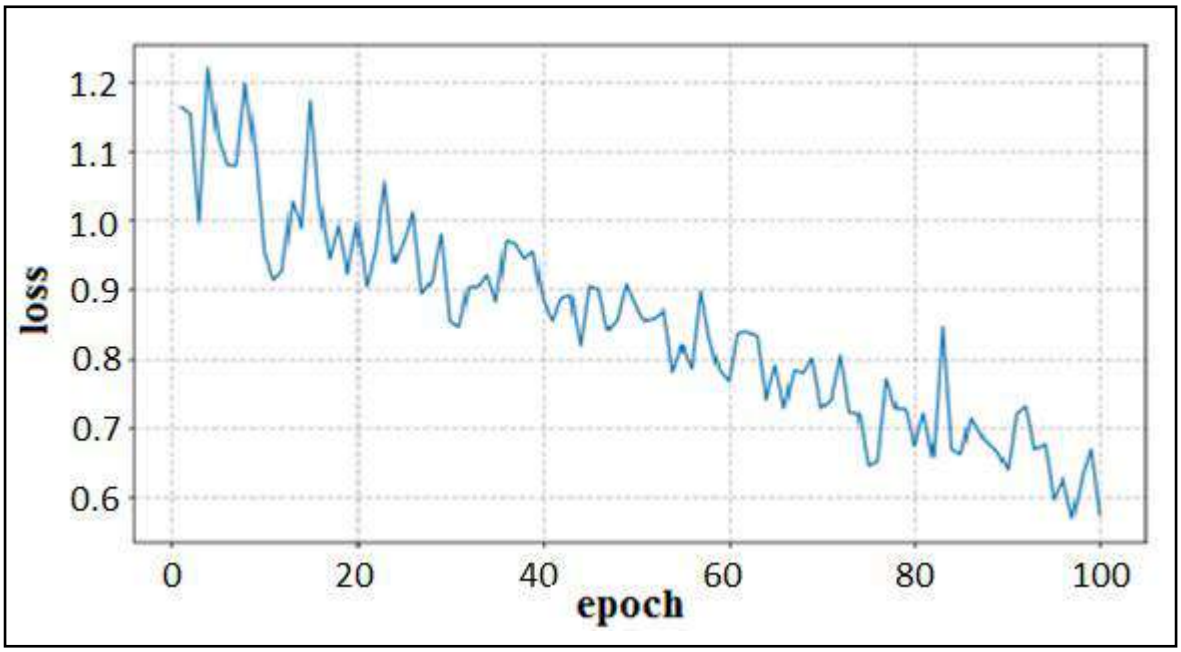

Рисунок 18 – График изменения loss при обучении
на сегментированных снимках

Вывод по главе 2

Как видно из таблицы 1, во всех экспериментах, кроме одного, при обучении модели с архитектурой DenseNet169, в котором значение осталось неизменным, можно наблюдать увеличение значения максимума метрики accuracy, что свидетельствует о росте точности предсказаний тестировавшихся предобученных моделей из раздела библиотеки keras – Keras Applications. Уменьшения значения минимального loss по результатам проведенных проверок не произошло, но и значительного роста в ходе тестов выявлено не было.

Также дополнительно было проведено трансферное обучение модели архитектуры DenseNet121 на 504 рентгенологических снимках в течение 100 эпох. Таким образом, при обучении на большем объеме данных и в течение большего количества эпох наблюдается незначительное увеличение метрики accuracy. График функции потерь при этом менее плавный, что делает обучение менее предсказуемым и может отрицательно отразится на конечной точности классификации модели.

Таким образом, можно предположить, что предобработка в виде семантической сегментации легких, может положительно повлиять на последующую классификацию, но эффект будет достаточно незначительным.

3. РЕАЛИЗАЦИЯ СЛИЯНИЯ МОДАЛЬНОСТЕЙ

Для обработки данных обеих модальностей: рентгенологических снимков и клинических данных, было использовано трансферное обучение. В обоих случаях использовались предобученные модели на основе архитектуры трансформер.

3.1. Архитектура transformers

Оригинальная архитектура трансформер была описана в статье «Attention Is All You Need» [35]. Общая схема архитектуры из статьи приведена на рисунке 19.

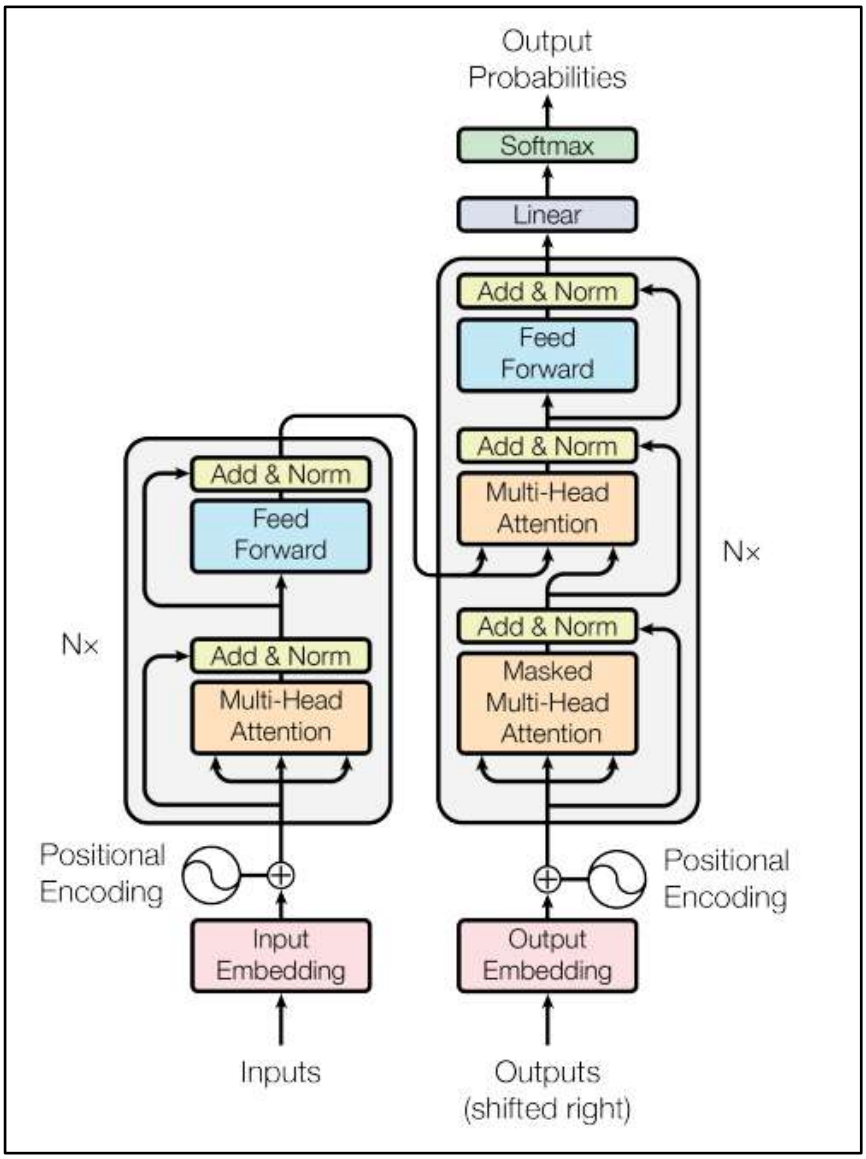

Рисунок 19 – Архитектура сети transformer [35]

Модель трансформера состоит из двух основных частей энкодера и декодера. Энкодер преобразует входные данные в эмбеддинги, которые потом передаются в декодер. Эмбеддинги представляют собой вектора, которые сопоставляют входную информацию с точкой в векторном пространстве [36]. Декодер на основе эмбеддингов из декодера и изначальных входных данных генерирует эмбеддинги, которые конвертируются в данные предсказания модели. В процессе обучения выходные эмбеддинги должны стремиться к тому, чтобы их значения были заполнены в соответствии с правилом – чем меньше расстояние между эмбеддингами в векторном пространстве, тем более схожи входные данные.

Преимуществом моделей трансформеров является наличие механизма внимания. За счет данного механизма информации из более ранних токенов не «забывается», как это происходит в RNN сетях. Также, в отличие от последних, в трансформерах, токены обрабатываются одновременно, а не последовательно. Что с одной стороны позволяет распараллелить процесс, но с другой приводит к потере информации о положении эмбеддинга в исходной последовательности. Для компенсации данного недостатка перед подачей данных в энкодер и декодер эмбеддинги конкатенируются с позиционными эмбеддингами. Позиционные эмбеддинги создаются с помощью переодических синусоидальных и косинусоидальных функций с изменяемой частотой.

Энкодер состоит из некоторого числа последовательно соединенных блоков-энкодеров одинаковой структуры. В них данные проходят через блок Multi-Head Attention и полносвязный слой. После каждого из последних производится сложение с тензорами перед входом в блок и нормировка.

Multi-Head Attention блок введен для того, чтобы модель учитывала контекст получаемых данных. Появление данного блока является развитием матрицы attention, когда на вход декодера подаются все скрытые состояния. На вход блока Multi-Head Attention подаются три матрицы: запро-

сов (Query), ключей (Key) и значений (Value). В блоке они перемножаются в соответствии с формулой (1):

$$\text{Attention}(Q, K, V)=\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \quad (1)$$

где Q – матрица запросов;

K – матрица ключей;

V – матрица значений;

d_k – размерность матриц ключей и значений.

После перемножения произведение делится на корень из размерности матриц для масштабирования. И к полученному значению применяется функция softmax. Перемножение матриц Query и Key производит линейное преобразование векторного пространства. На рисунке 20 приведена схема обработки данных в механизме attention.

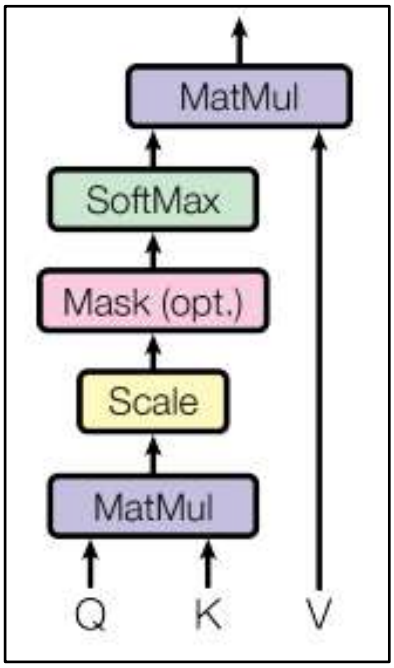

Рисунок 20 – Масштабированное скалярное произведение внимания [35]

В ходе обучения данные матрицы корректируются так, чтобы максимально удалить друг от друга точки, которые принадлежат к разным группам в решаемой задаче. Для получения предсказания производится

умножение на матрицу Value. Описанная операция называется self attention.

В Multi-Head Attention выполняется одновременно несколько self attention. Эти операции принято называть heads. Параллельное вычисление нескольких attention увеличивает точность модели. Инициализации матриц разными случайными числами достаточно, чтобы оптимизация привела к различным результатам. Таким образом, создается несколько потенциально эффективных решений. Перед выходом из блока Multi-Head Attention полученные матрицы конкатенируются, после чего понижается их размерность. На рисунке 21 изображена схема блока Multi-Head Attention. В ней слои heads обозначены буквой h.

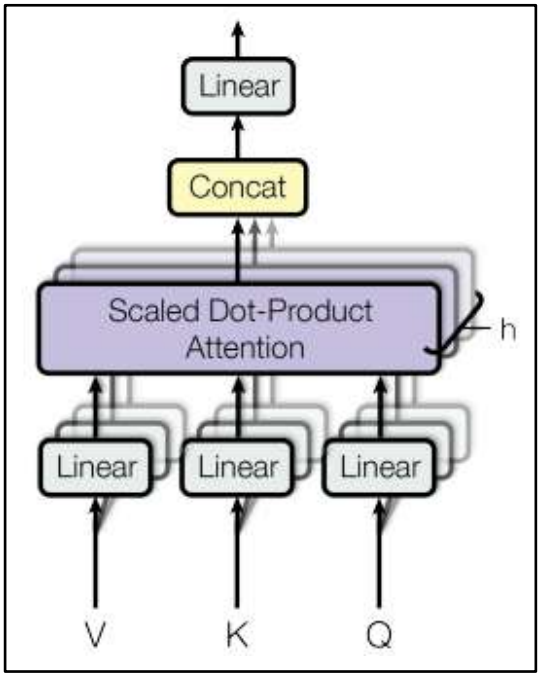

Рисунок 21 – Multi-Head Attention блок [35]

Выходные матрицы attention возможно визуализировать. На рисунке 22 приведены результаты применения функции визуализации матрицы внимания при обучении модели на рентгенологических снимках.

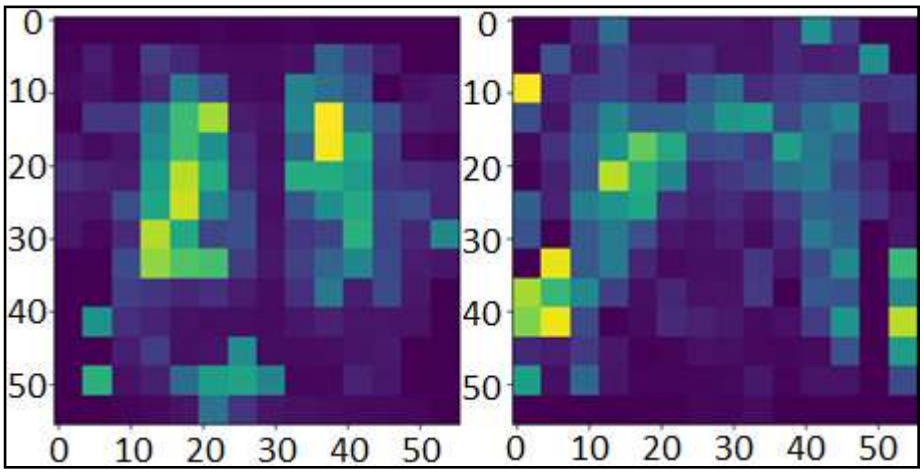

Рисунок 22 – Визуализация матриц attention

В листинге 2 приведена функция для сохранения визуализированных матриц в формате png. С ее помощью были получены изображения на рисунке 22.

Листинг 2 – Функция визуализации матриц attention

```
def attention_img(outputs, pixel_val):
    attentions = outputs.attentions[-1]
    nh = attentions.shape[1]
    attentions = attentions[0, :, 0, 1:].reshape(nh, -1)
    pixel_val = torch.tensor(pixel_val[0])
    threshold = 0.6
    w_featmap = pixel_val.shape[-2] // batch_size
    h_featmap = pixel_val.shape[-1] // batch_size
    w_featmap = 14
    h_featmap = 14
    val, idx = torch.sort(attentions)
    val /= torch.sum(val, dim=1, keepdim=True)
    cumval = torch.cumsum(val, dim=1)
    th_attn = cumval > (1 - threshold)
    idx2 = torch.argsort(idx)
    for head in range(nh):
        th_attn[head] = th_attn[head][idx2[head]]
    th_attn = th_attn.reshape(nh, w_featmap, h_featmap).float()
    th_attn = nn.functional.interpolate(th_attn.unsqueeze(0),
                                        scale_factor=batch_size,
                                        mode="nearest")[0].cpu().numpy()
    attentions = attentions.reshape(nh, w_featmap, h_featmap)
    attentions = nn.functional.interpolate(attentions.unsqueeze(0),
                                           scale_factor=batch_size,
                                           mode="nearest")[0].cpu()
    attentions = attentions.detach().numpy()
    output_dir = '.'
    os.makedirs(output_dir, exist_ok=True)
    for j in range(nh):
        fname = os.path.join(output_dir, "attn-head" + str(j) + ".png")
        plt.figure()
        plt.imshow(attentions[j])
        plt.imsave(fname=fname, arr=attentions[j], format='png')
```

Как можно заметить в областях, где на снимках располагаются легкие более высокие значения внимания. Декодер также состоит из повторяющихся блоков, но с добавлением вначале Masked Multi-Head Attention блока. Masked Multi-Head Attention блок отличается от Multi-Head Attention блока тем, что в нем скрывается часть данных при обучении. Поскольку декодер является авторегрессионным и генерирует последовательность слово за словом, необходимо предотвращать вычисление оценок внимания для будущих токенов [37].

Кроме описанного варианта с энкодером и декодером возможно создание моделей только с энкодером, например, к ним относится BERT, и только с декодером как GPT. Модели с декодером в основном используются для задач генерации.

3.2. Классификация модальности рентгенологических снимков

Для части модели, обрабатывающей рентгенологические снимки использована вариация архитектуры трансформер – Vision Transformer (ViT) [38]. Данная архитектура разработана для применения технологии обработки из трансформеров на графических данных.

В ViT используется только энкодер. В целом архитектура энкодера ViT аналогична стандартной архитектуре энкодера трансформера за исключением начальных этапов. На вход модели ViT вместо слов подается разбитое на части изображение, а не слова. Части изображения принято называть патчами. Посредством пропуска их через полносвязный слой генерируются эмбеддинги. В итоге получаются векторы для каждого патча. Также добавляется дополнительный вектор для получения общего представления о поступившем на вход изображении. Схема архитектуры модели ViT изображена на рисунке 23.

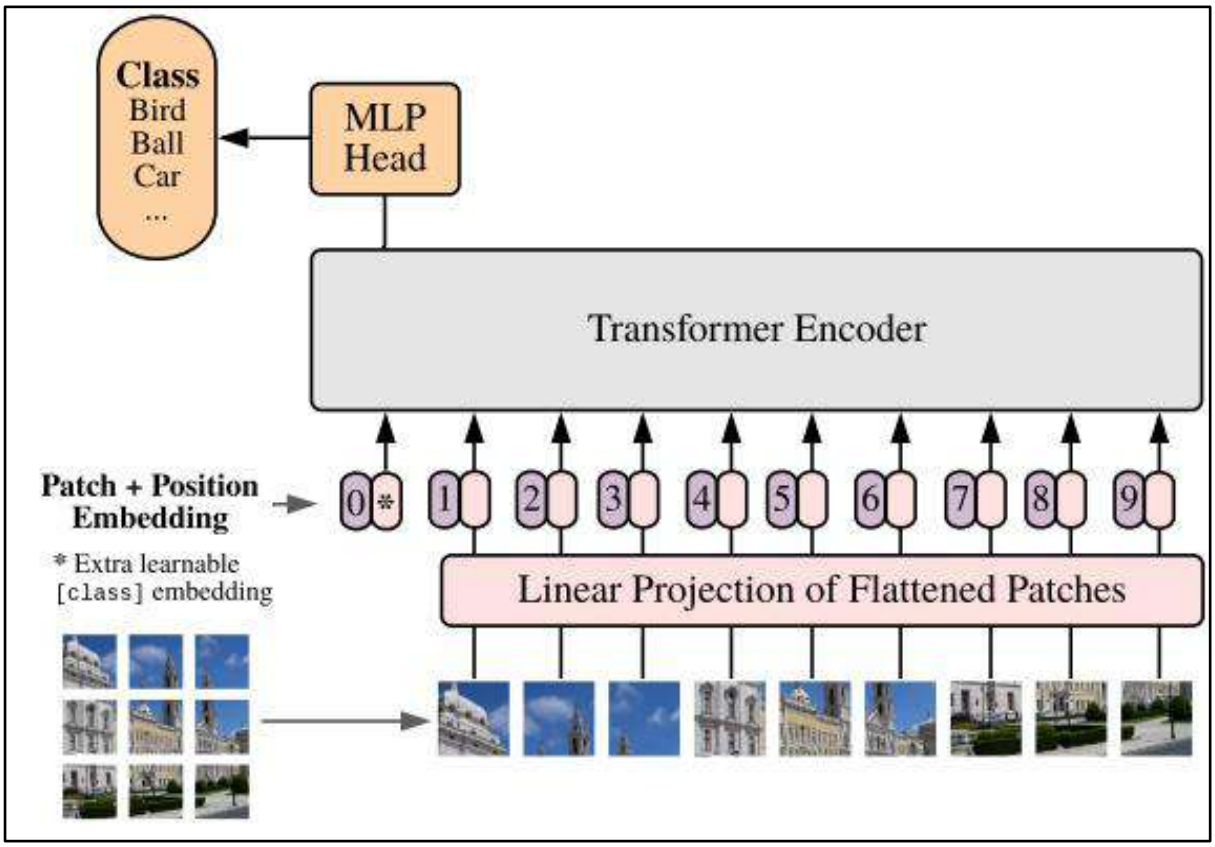

Рисунок 23 – Архитектура сети ViT [38]

Поскольку эмбеддинги патчей, как и в оригинальной архитектуре трансформеров обрабатываются параллельно, возникает необходимость восполнить потерю информации о позициях, из которых был взят тот или иной патч. Для этого вводятся позиционные эмбеддинги.

Предварительно обученные модели содержат внутреннее представление изображений, которое затем можно использовать для извлечения функций, полезных для задач отличных от тех на которых проходило предобучение. В связи с этим, для создания модели классификации данных модальности рентгенологических снимков, за основу модели выбрана предобученная модель, которая имеет архитектуру Vision Transformer (ViT).

Непосредственно для реализации модели для классификации рентгенологических снимков была использована предобученная на наборе данных ImageNet-21k, модель vit-base-patch16-224-in21k от компании Google [39]. При предобучении vit-base-patch16-224-in21k было использо-

вано 14 миллионов изображений, распределенных на 21843 класса. В листинге 3 приведен код с описанием класса модели для классификации изображений.

Листинг 3 – Класс модели для классификации изображений

```
from transformers import ViTImageProcessor, ViTModel
from transformers.modeling_outputs import SequenceClassifierOutput

VIT = ViTModel.from_pretrained('google/vit-base-patch16-224-in21k')
class ViTForImageClassification(nn.Module):
    def __init__(self, head, num_labels=MODEL_NUM_LABELS):
        super(ViTForImageClassification, self).__init__()
        self.vit = VIT
        self.dropout = nn.Dropout(0.5)
        self.head = head
    def forward(self, pixel_values):
        pixel_values_0 = torch.tensor(pixel_values).clone().detach()
        outputs = self.vit(pixel_values=pixel_values_0)
        last_hidden_states = outputs.pooler_output
        output = self.dropout(last_hidden_states)
        return self.head(output)
```

Класс имеет поля `vit`, `dropout` и `head`. В поле `vit` хранится предобученная модель ViT. В поле `dropout` хранится слой dropout с 10 % вероятностью обнуления. Поле `head` содержит `nn.Sequential` слой, преобразующий вектор выходных данных из модели ViT в вектор размерности равной количеству классов, на которые производится классификация.

В листинге 4 приведен код непосредственно создания модели для классификации рентгенологических снимков.

Листинг 4 – Создание модели для классификации рентгенологических снимков

```
MODEL_NUM_LABELS = 2
EMBEDDING_DIM = 768

img_head = nn.Sequential(
        nn.Linear(EMBEDDING_DIM, MODEL_NUM_LABELS)).to(device)
model_img = ViTForImageClassification(img_head)
```

На вход модель принимает изображения 224 на 224 пикселя. Размерность векторов эмбеддингов составляет 768. Размер фрагментов, на которые разбивается изображение – 16 пикселей. В архитектуре использовано 12 блоков энкодера ViT.

Для обучения использован оптимизатор Adam из torch.optim. В качестве функции потерь выбрана CrossEntropyLoss [40].

3.3. Классификация модальности клинических данных

Для классификации клинических данных в виде медицинских выводов по изображениям использована модель на основе предобученной модели BERT. Конкретно использована bert-base-uncased содержащая 110M параметров. Для обучения использовано 515 текстов. Пример образца классифицируемых клинических данных приведен на рисунке 24.

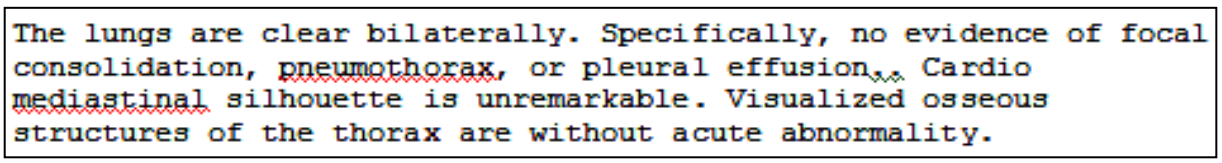

Рисунок 24 – Пример образца клинических данных

BERT относится к моделям-трансформерам. Данная модель была разработана компанией Google [41]. Традиционно она используется для решения NLP-задач, таких как перевод, генерация ответов и так далее. Отличительной особенностью BERT является то, что она учитывает контекст текста как слева направо, так и в обратном направлении. Данный эффект был достигнут за счет предобучения на маскированной языковой модели. При таком подходе модель училась предсказывать не продолжения фраз, а слова, располагающиеся в середине текста.

В листинге 5 приведен код загрузки токенизатора BERT для перевода текста в токены и самой модели.

Листинг 5 – Подготовка предобученной модели BERT

```
TEXT_MODEL_CARD = 'bert-base-uncased'
tokenizer = BertTokenizer.from_pretrained(TEXT_MODEL_CARD)
embedder = BertModel.from_pretrained(TEXT_MODEL_CARD).to(device)

for params in embedder.parameters():
    params.requires_grad = False
```

В листинге 6 представлено описание класса модели для классификации медицинских записей.

Листинг 6 – Класс модели для классификации клинических данных

```
class TextModel(nn.Module):
    def __init__(self, tokenizer, embedder, head):
        super().__init__()
        self.tokenizer = tokenizer
        self.embedder = embedder
        self.head = head
    def forward(self, texts, audio_features):
        tokenizer_output = self.tokenizer(texts, return_tensors='pt',
                                          padding=True,
                                          truncation=False).to(device)
        embedder_output = self.embedder(**tokenizer_output,
                                        output_hidden_states=True)
        text_features = embedder_output['last_hidden_state']
        t_o_s = tokenizer_output.attention_mask.unsqueeze(-1)
        text_features_sum = (text_features * t_o_s).sum(axis=1)
        t_o_a = tokenizer_output.attention_mask.sum(axis=1).unsqueeze(-1)
        text_features_pooled = text_features_sum / t_o_a
        return self.head(text_features_pooled)
```

В листинге 7 приведено непосредственное создание экземпляра класса TextModel.

Листинг 7 – Создание экземпляра класса TextModel

```
text_only_head = nn.Sequential(
        nn.Linear(EMBEDDING_DIM, head_hidden_dimension),
        nn.ReLU(),
        nn.Linear(head_hidden_dimension, MODEL_NUM_LABELS)
    ).to(device)
    text_only = TextModel(tokenizer, embedder, text_only_head)
```

Матрицы весов BERT перед обучением были «заморожены». Тонкая настройка производится за счет обучения слоев, передаваемых в атрибут класса `head`. Для отслеживания точности выбрана метрика accuracy, функция потерь – Cross entropy loss. В качестве оптимизатора используется алгоритм Adam.

3.4. Классификация на основе двух модальностей

Модель для классификации на основе модальностей рентгенологических снимков и клинических данных создана с использованием ранее описанных предобученных моделей vit-base-patch16-224-in21k и bert-base-uncased. Для обмена эмбеддингами между трансформерами к оригиналь-

ным архитектурам были добавлены блоки MulTA_CrossAttentionBlock, обеспечивающие передачу матриц [42]. Описание класса MulTA_CrossAttentionBlock приведено в листинге 8.

Листинг 8 – Описание класса MulTA_CrossAttentionBlock

```
class MulTA_CrossAttentionBlock(nn.Module):
    def __init__(self, embedding_dim, d_ffn, num_heads=4,
dropout_prob=0.1):
        super().__init__()
        self.embedding_dim = embedding_dim
        self.d_ffn = d_ffn
        self.num_heads = num_heads
        self.dropout_prob = dropout_prob
        self.layer_norm = nn.LayerNorm(self.embedding_dim)
        self.mh_attention = nn.MultiheadAttention(
                    embed_dim=self.embedding_dim,
                    num_heads=self.num_heads,
                    dropout=self.dropout_prob,
                    batch_first=True)
        self.pointwise_ff = PositionwiseFeedForward(
                    d_model=self.embedding_dim,
                    hidden=self.d_ffn)
```

Данный класс описывает блок, который практически идентичен классическим блокам трансформеров. За исключением того, что в экземпляры класса MulTA_CrossAttentionBlock передаются матрицы ключей и значений из сторонней модальности и матрица вопросов изначальной модальности. Как и в обычных блоках к матрицам применяется Multi-Head Attention результат, которого складывается с матрицей вопросов. После этого данные проходят через слои нормализации и Feed Forward. Слой Feed Forward в свою очередь состоит из двух полносвязных слоев с ReLu и dropout. На рисунке 25 приведена схема обмена матрицами вопросов, ключей и значений между отдельными блоками трансформеров.

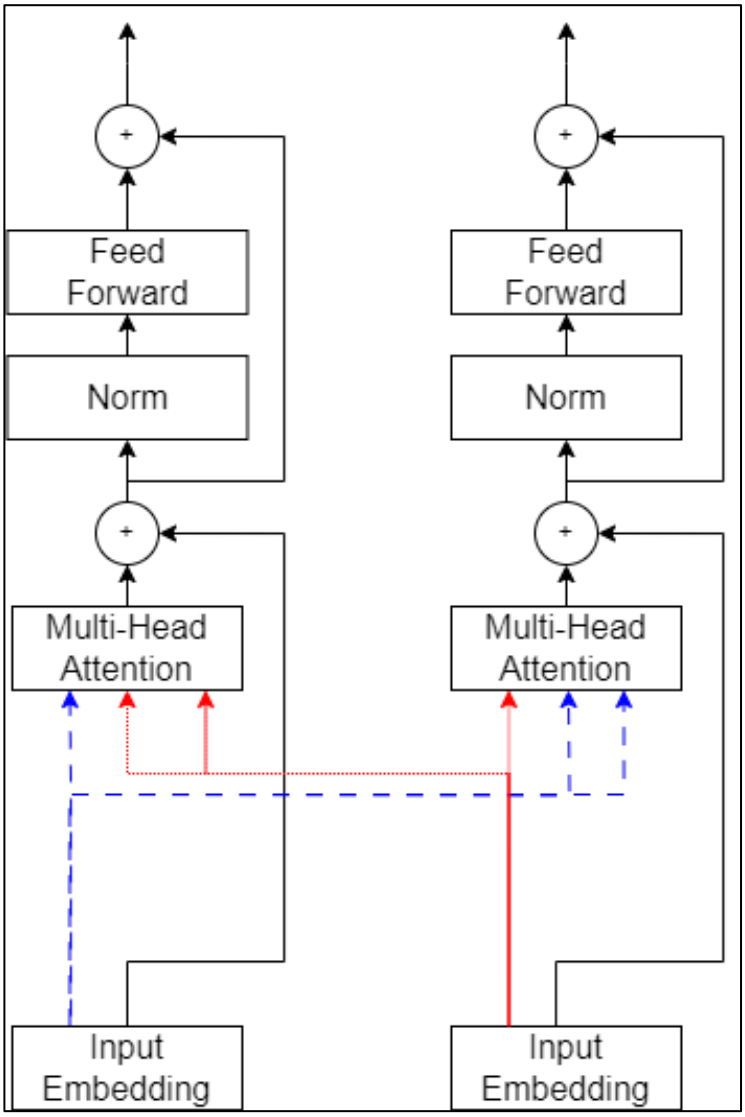

Рисунок 25 – Схема блока трансформера с обменом эмбеддингами

Выходные тензоры последних блоков трансформеров конкатенируются и передаются в общие слои, описанные в поле `head`. В созданной реализации общие слои представляют собой три полносвязных слоя с ReLu размерностями 768 на 224, 224 на 56 и 56 на 2.

Для обучения использован оптимизатор Adam. В качестве метрики используется accuracy, функция потерь – CrossEntropyLoss.

Описание класса MulTA, описывающего мультимодальную модель, приведено на листинге 2 в приложении Б. Код создания экземпляра класса MulTA приведен в листинге 9.

Листинг 9 – Создание экземпляра класса MulTA

```
multa_d_ffn=2048
hidden_state_index=8
multa_nblocks=4
head_hidden_dimension=224
dropout_prob=0.1
unaligned_head = nn.Sequential(
        nn.Dropout(0.1),
        nn.Linear(EMBEDDING_DIM*2, head_hidden_dimension),nn.ReLU(),
        nn.Linear(head_hidden_dimension, head_hidden_dimension//4),
        nn.ReLU(),
        nn.Linear(head_hidden_dimension//4, MODEL_NUM_LABELS)
    ).to(device)
unaligned_mm_model = MulTA(embedding_dim=EMBEDDING_DIM,
       d_ffn=multa_d_ffn,
       n_blocks=multa_nblocks,
       head=unaligned_head,
       hidden_state_index=hidden_state_index,
       dropout_prob=dropout_prob
       ).to(device)
```

Вывод по главе 3

В данной главе были описаны три созданные в рамках выполнения данной работы модели:

– модель для классификации рентгенологических снимков;

– модель для классификации клинических данных;

– модель для классификации на основе рентгенологических снимков и клинических данных.

Для создания всех трех моделей был использован метод трансферного обучения. В основе архитектур моделей лежит архитектура трансформер.

4. АНАЛИЗ СОЗДАННЫХ МОДЕЛЕЙ

В данной главе будет проведен анализ качества полученных моделей на основе графиков изменения метрик в процессе их обучения.

4.1. Модель модальности рентгенологических снимков

Графики изменения accuracy и loss при обучении модели классификации рентгенологических снимков на данных, подвергшихся сегментированию, приведены на рисунках 26–27.

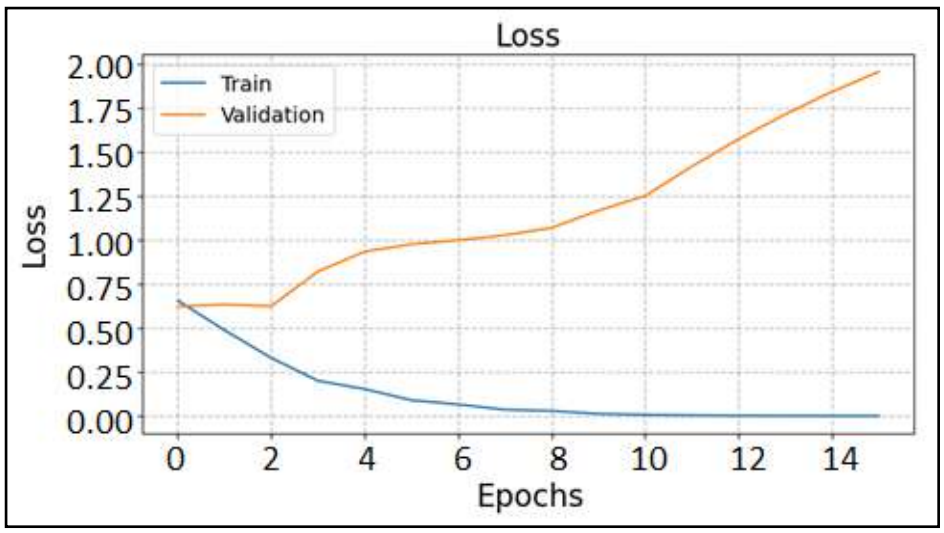

Рисунок 26 – Изменение loss при обучении на сегментированных снимках

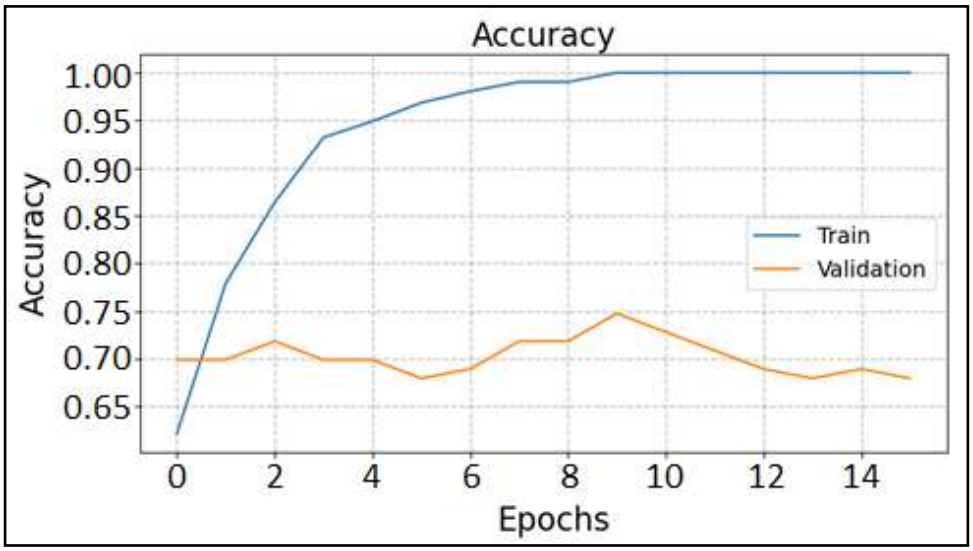

Рисунок 27 – Изменение accuracy при обучении
на сегментированных снимках

Графики изменения accuracy и loss при обучении на несегментированных данных приведены на рисунках 28–29.

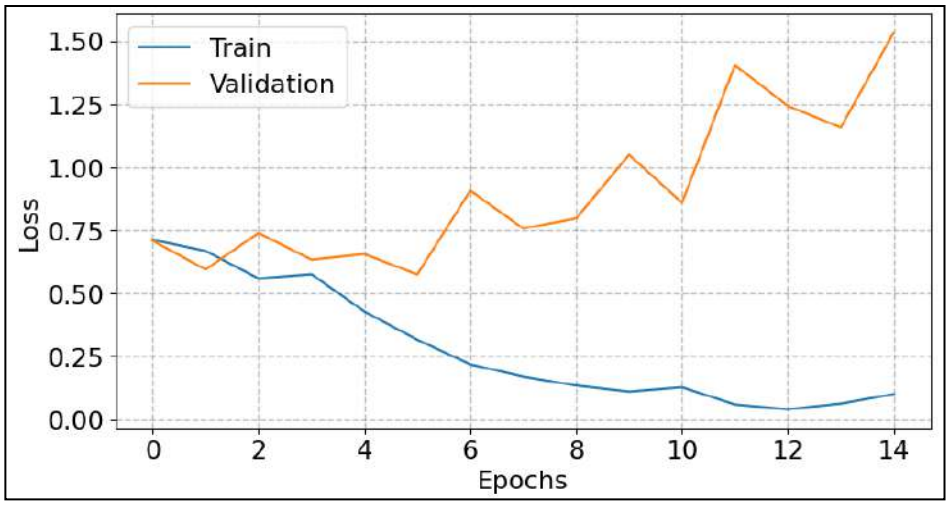

Рисунок 28 – Изменение loss при обучении
на несегментированных снимках

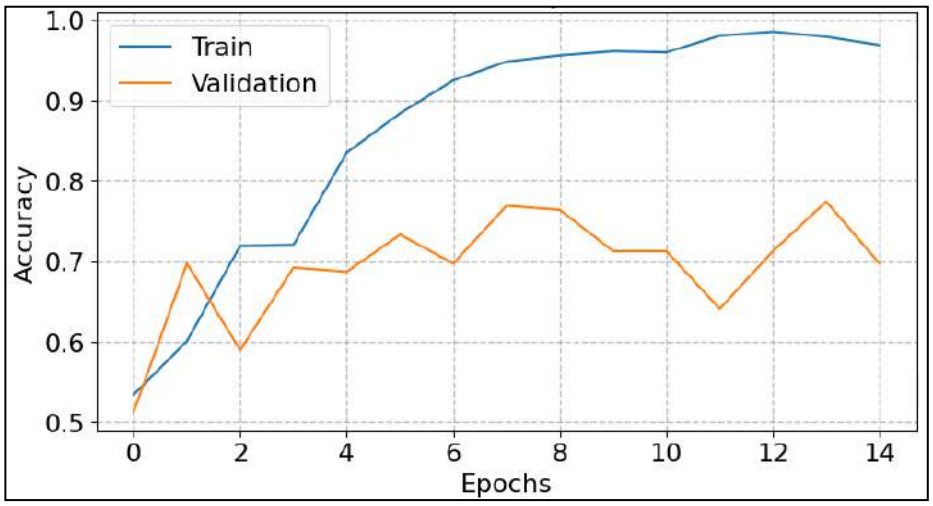

Рисунок 29 – Изменение accuracy при обучении
на несегментированных снимках

Как видно на графиках в обоих случаях модель начала переобучаться уже на первых эпохах. При этом при использовании несегментированных снимков обучение прошло немного лучше. Значения loss меньше, а accuracy выше. Наилучшая точность в 77% также получена на необработанных данных.

4.2. Модель модальности клинических данных

Графики изменения accuracy и loss при обучении модели для классификации текстов с описанием клинических случаев приведены на рисунках 30–31.

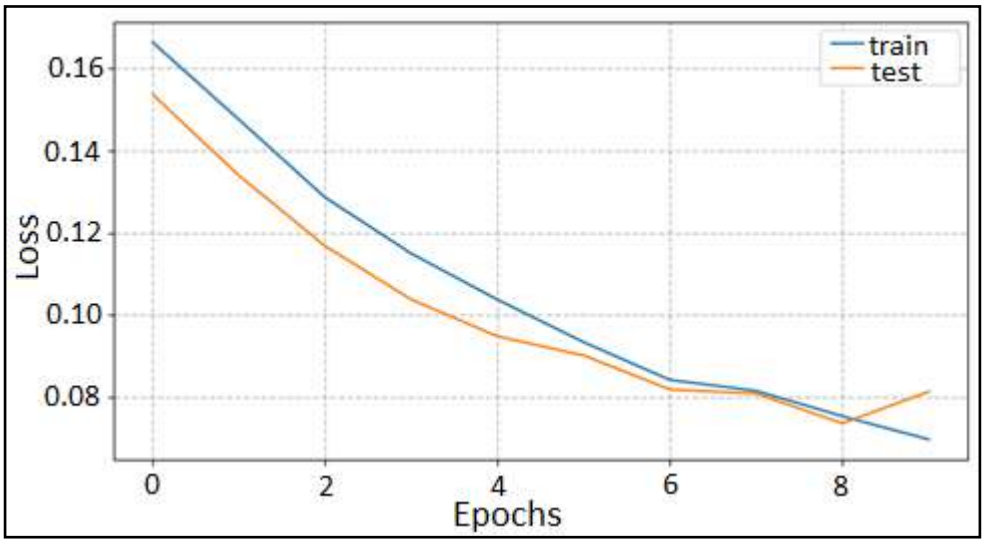

Рисунок 30 – Изменение loss при обучении модели классификации текстов

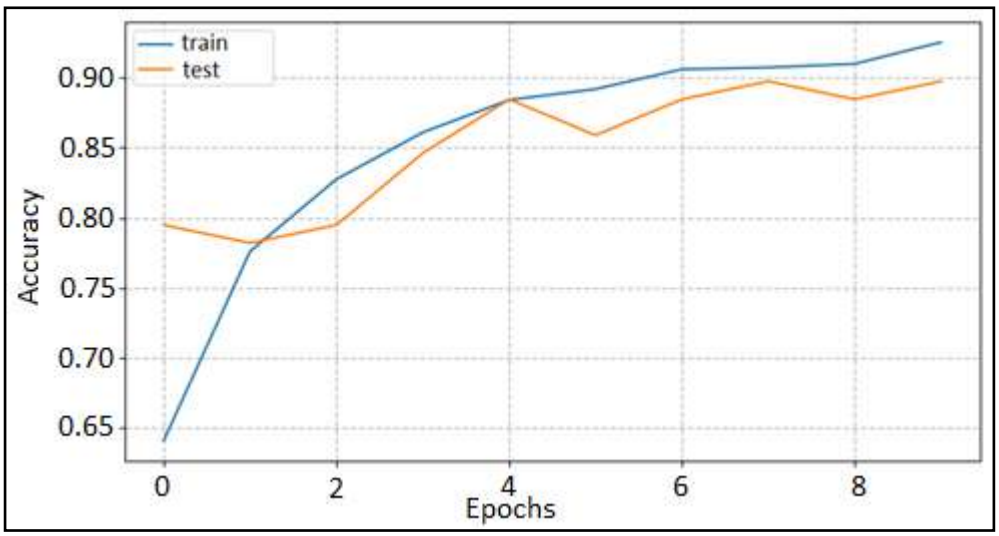

Рисунок 31 – Изменение accuracy при обучении
модели классификации текстов

В ходе обучения была достигнута достаточно высокая точность – 88%. Такой результат, прежде всего можно объяснить спецификой обучающих данных. Описания состоят из кратких тезисов с небольшими ва-

риациями формулировок, что делает их классификацию достаточно простой задачей.

4.3. Классификация на основе двух модальностей

Графики обучения модели на наборах данных с использованием сегментированных снимков проведены на рисунках 32–33.

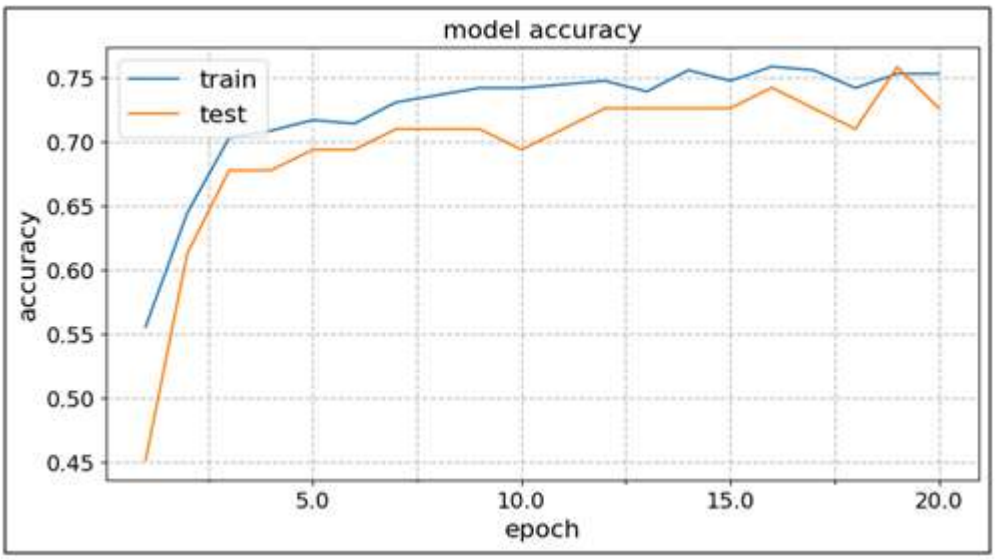

Рисунок 32 – Изменение accuracy при обучении мультимодальной модели с использованием сегментированных снимков

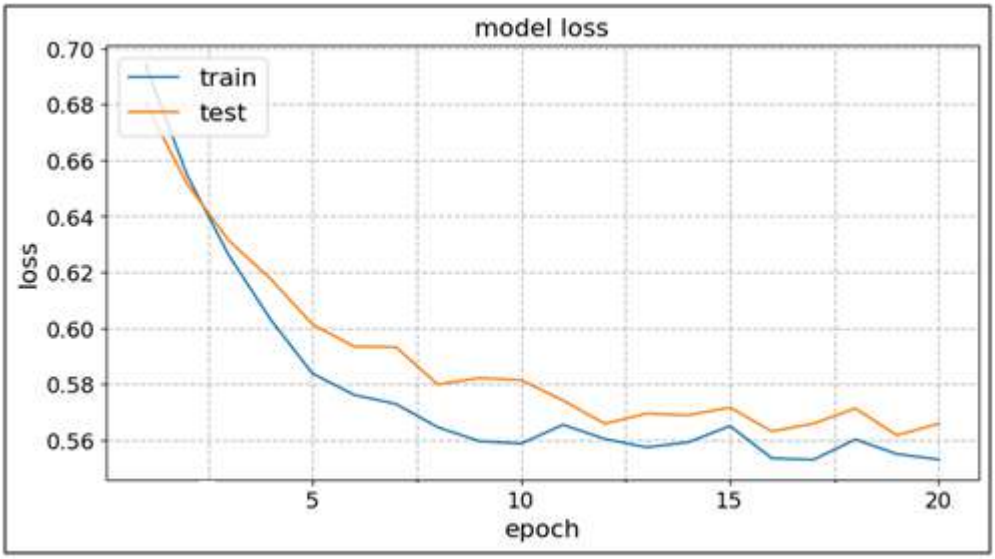

Рисунок 33 – Изменение accuracy при обучении мультимодальной модели с использованием сегментированных снимков

Графики обучения модели на наборах данных с использованием несегментированных проведены на рисунках 34–35.

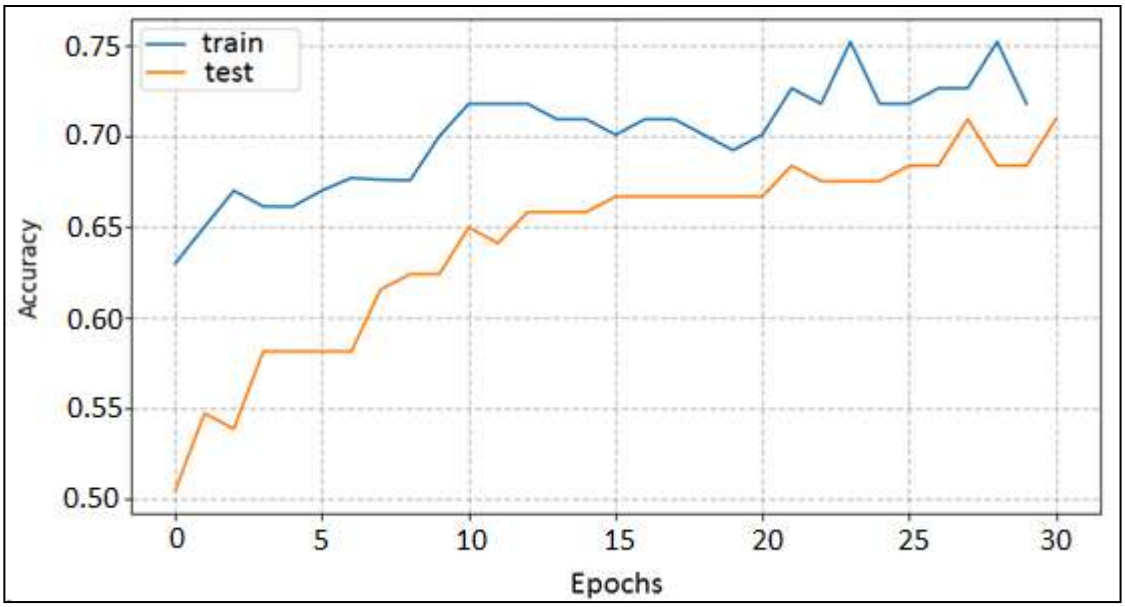

Рисунок 34 – Изменение accuracy при обучении мультимодальной модели с использованием несегментированных снимков

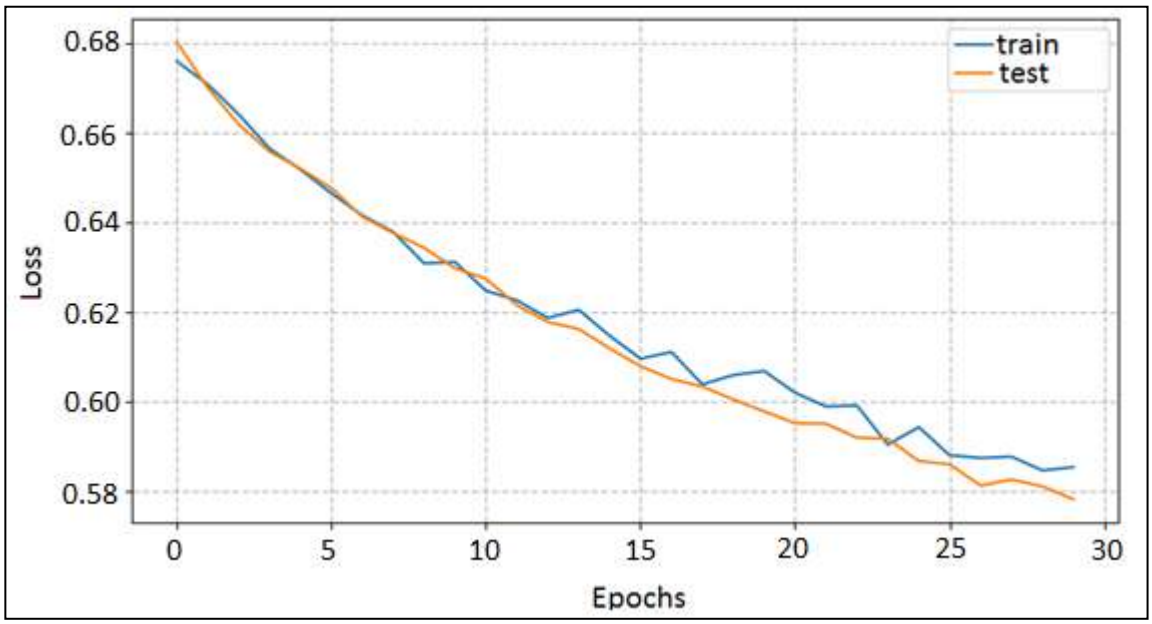

Рисунок 35 – Изменение loss при обучении мультимодальной модели с использованием несегментированных снимков

Как видно на графиках обучении с использованием сегментации для приведенных случаев оказалось более эффективным. Валидационное зна-

чение accuracy имеет большее значение и меньший разрыв со значениями тренировочной выборки. Также наилучшее значение в 75% достигается на сегментированных данных. В связи с этим было проведено обучение с использованием сегментированных снимков на большем количестве эпох. Графики accuracy и loss представлены на рисунках 36–37.

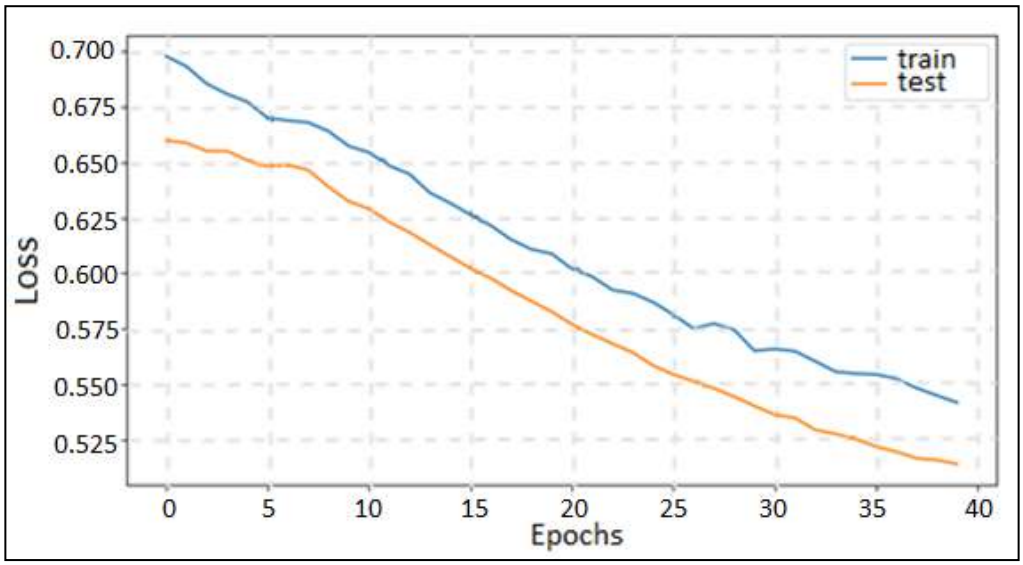

Рисунок 36 – Изменение loss при обучении мультимодальной модели с использованием сегментированных снимков на 40 эпохах

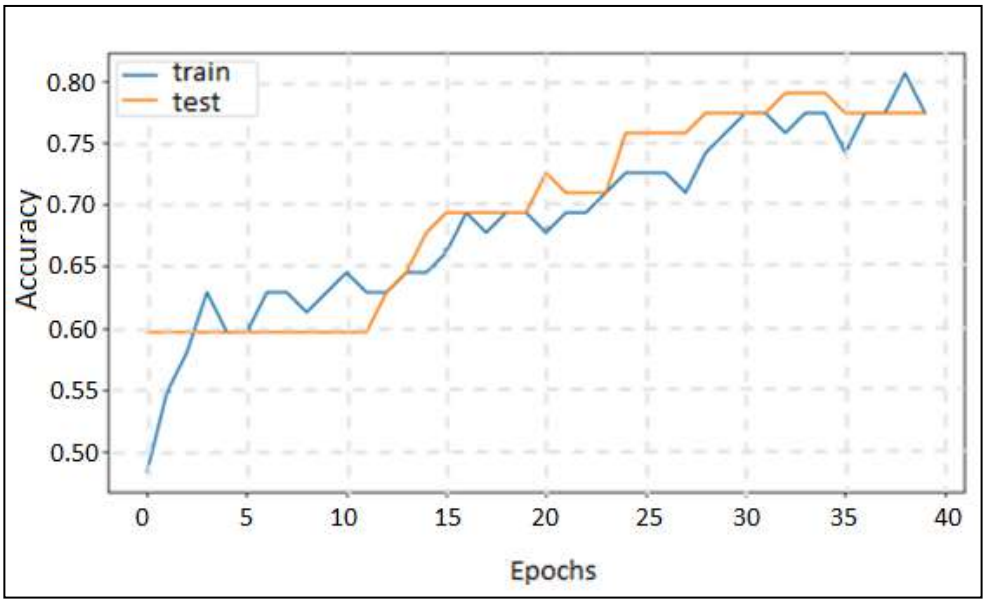

Рисунок 37 – Изменение accuracy при обучении мультимодальной модели с использованием сегментированных снимков на 40 эпохах

В результате обучения максимальное значение accuracy равняется 79 процентам, достигаемым на 32 эпохе обучения. При этом модель не переобучена и график accuracy не вышел на плато, что может свидетельствовать о потенциальном увеличении точности при дальнейшем обучении.

Вывод по главе 4

При измерении точности классификации метрикой accuracy максимальные значения, которые были получены: 77 процентов на классификации рентгенологических снимков, 88 процентов на классификации клинических данных, 79 процентов на классификации на основе обеих модальностей. Таким образом, лучшую точность в соответствии с выбранной метрикой показало применение модели классификации модальности клинических данных.

Точность мультимодальной модели оказалась выше модели классификации рентгенологических снимков, но ниже модели классификации клинических данных. Что можно объяснить тем, что результат ее классификации был промежуточным между двумя другими моделями. Поскольку точность одномодальных моделей отличается почти на 10 процентов, вместо взаимной корректировки, в данном случае, имел место обратный эффект. Менее качественная модель внесла помехи в предсказания более успешной модели.

ЗАКЛЮЧЕНИЕ

В рамках данной работы была разработана модель промежуточного слияния модальности рентгенологических снимков с мэппингом эмбеддингов из модальности клинических данных для классификации заболеваний легких. При этом были решены следующие задачи.

1. Проведена сегментация снимков легких из набора данных для обучения.
2. Проведено обучение модели промежуточного слияния модальности рентгенологических снимков на наборе данных из сегментированных и несегментированных рентгенологических снимков.
3. Проведено сравнение точности модели при обучении на сегментированных и несегментированных рентгенологических снимках.
4. Произведен мэппинг эмбеддингов из модальности клинических данных.
5. Проанализированы изменения точности модели после мэппинга эмбеддингов модальности клинических данных.

Файлы с полным кодом описываемом в данной работе хранятся в репозитории github [43].

ЛИТЕРАТУРА

1. Лобанцев А.А. Методы и алгоритмы обработки мультимодальных медицинских данных на основе переноса знаний в системах поддержки принятия клинических решений: дис. … канд. техн. наук: 05.13.10: защищена 21.12.2020 г. – М., 2020. – 218 с.

2. Златорев А.М. Сравнительная оценка эффективности диагностики заболеваний легких на основе дискриминационного анализа и нейронных сетей. / А.М. Элаторев, Н.Е. Казимирова, М.А. Радченко // Вестник СГТУ, 2012. – № 1 (64). – 322–327 с.

3. Уткин Л.В. Медицинские интеллектуальные системы на примере диагностики рака легкого. / Л.В. Уткин, А.А. Мелдо, О.С. Ипатов, М.А. Рябинини // Известия ЮФУ, Технические науки, 2018. – 9 с.

4. Сперанская А.А. Роль искусственного интеллекта в оценке прогрессирующих фиброзирующих болезней легких. // Терапевтический архив, 2022. – № 94 (3). – С. 409–412.

5. De Vries R. Prediction of response to anti-PD-1 therapy in patients with non-small-cell lung cancer by electronic nose analysis of exhaled breath. / R. de Vries, M. Muller, V. Van der Noort. // Annals of Oncology, 2019. – v. 30. №10 – 1660–1666 pp.

6. Aljbawi W., Simmons S.O., Urovi V. Developing a Multi-variate Prediction Model For The Detection of COVID-19 From Crowd-sourced Respiratory Voice Data [Электронный ресурс] // arXiv.org, 2022. Дата обновления: 08.09.2022 г. URL: https://arxiv.org/abs/2209.03727 (дата обращения: 12.05.2024 г.).

7. Audebert N., Herold C., Slimani K., Vidal C. Multimodal deep networks for text and image-based document classification // arXiv.org, 2019. Дата обновления: 15.07.2019 г. URL: https://arxiv.org/abs/1907.06370 (дата обращения: 12.05.2024 г.).

8. Kumar G.K., Nandakumar K. Hate-CLIPper: Multimodal Hateful Meme Classification based on Cross-modal Interaction of CLIP Features //

arXiv.org, 2022. Дата обновления: 17.10.2022 г. URL: https://arxiv.org/abs/2210.05916 (дата обращения: 12.05.2024 г.).

9. Лобанцев А.А. Система поддержки клинических решений с обработкой мультимодальных медицинских данных как средство повышения эффективности работы врача-радиолога. // Научно-технический вестник информационных технологий, механики и оптики, 2020. – т. 20, №6. – С. 893–897.

10. Mallya M., Hamarneh G. Deep Multimodal Guidance for Medical Image Classification // arXiv.org, 2022. Дата обновления: 21.07.2022 г. URL: https://arxiv.org/abs/2203.05683 (дата обращения: 12.05.2024 г.).

11. Лакшманан В. Машинное обучение. Паттерны проектирования. / В. Лакшманан, С. Робинсон, М. Мунн; пер. с. анг. // СПб.: БХВ-Петербург, 2022. – 448 с.

12. Alaba S.Y., Gurbuz A.C., Ball J.E. A Comprehensive Survey of Deep Learning Multisensor Fusion-based 3D Object Detection for Autonomous Driving: Methods, Challenges, Open Issues, and Future Directions. // IEEE Transactions on intelligent transportation systems, 2023. – 11 p.

13. Joze H.R.V., Shaban A., Iuzzolino M.L., Koishida K. MMTM: Multimodal Transfer Module for CNN Fusion // arXiv.org, 2019. Дата обновления: 30.03.2020 г. URL: https://arxiv.org/abs/1911.08670 (дата обращения: 12.05.2024 г.).

14. Cangea C., Velickovic P., Lio P. XFlow: Cross-modal Deep Neural Networks for Audiovisual Classification // arXiv.org, 2019. Дата обновления: 12.04.2019 г. URL: https://arxiv.org/pdf/1709.00572 (дата обращения: 12.05.2024 г.).

15. Boulahia S.Y. Early, intermediate and late fusion strategies for robust deep learning-based multimodal action recognition. / S.Y. Boulahia, A. Amamra, M.R. Madi, S. Daikh – 2021. – 17 p.

16. Zheng L., Yang Y., Tian Q. SIFT Meets CNN: A Decade Survey of Instance Retrieval // arXiv.org, 2017. Дата обновления: 23.05.2017 г. URL: https://arxiv.org/abs/1608.01807 (дата обращения: 12.05.2024 г.).

17. Вижу, значит существую: обзор Deep Learning в Computer Vision (часть 2). [Электронный ресурс] URL: https://habr.com/ru/company/mipt/blog/458190/ (дата обращения: 15.05.2024 г.).

18. Mask R-CNN: архитектура современной нейронной сети для сегментации объектов на изображениях. [Электронный ресурс] URL: https://habr.com/ru/post/421299/ (дата обращения: 26.02.2024 г.).

19. Ronneberger O. U-Net: Convolutional Networks for Biomedical Image Segmentation. / O. Ronneberger, P. Fischer, T. Brox. – Germany: University of Freiburg. – 8 с.

20. Chest X-rays (Indiana University). [Электронный ресурс] URL: https://www.kaggle.com/datasets/raddar/chest-xrays-indiana-university?select=indiana_reports.csv (дата обращения: 02.05.2024 г.).

21. Lung segmentation from Chest X-Ray dataset. [Электронный ресурс] URL: https://clck.ru/37FURw (дата обращения: 26.02.2024 г.).

22. Chest Xray Masks and Labels. [Электронный ресурс] URL: https://www.kaggle.com/datasets/nikhilpandey360/chest-xray-masks-and-labels (дата обращения: 26.02.2024 г.).

23. TransferLearning and Unet to segment rocks on moon. [Электронный ресурс] URL: https://www.kaggle.com/code/basu369victor/transferlearning-and-unet-to-segment-rocks-on-moon/notebook (дата обращения: 02.05.2024 г.).

24. Pulmonary Chest X-Ray Abnormalities. [Электронный ресурс] URL: https://www.kaggle.com/datasets/kmader/pulmonary-chest-xray-abnormalities?select=ChinaSet_AllFiles (дата обращения: 02.05.2024 г.).

25. Lung Masks for Shenzhen Hospital Chest X-ray Set. [Электронный ресурс] URL: https://www.kaggle.com/datasets/yoctoman/shcxr-lung-mask (дата обращения: 02.05.2024 г.).

26. TensorFlow tf.keras.metrics.BinaryAccuracy. [Электронный ресурс] URL: tensorflow.org/api_docs/python/tf/keras/metrics/BinaryAccuracy (дата обращения: 02.05.2024 г.).

27. Dice Coefficient! What is it? [Электронный ресурс] URL: https://medium.com/@lathashreeh/dice-coefficient-what-is-it-ff090ec97bda (дата обращения: 02.05.2024 г.).

28. Keras Applications. [Электронный ресурс] URL: https://keras.io/api/applications/ (дата обращения: 26.02.2024 г.).

29. Huang G., Liu Z., Maaten L. Densely Connected Convolutional Networks // arXiv.org, 2016. Дата обновления: 28.01.2018 г. URL: https://arxiv.org/abs/1608.06993 (дата обращения: 12.05.2024 г.).

30. Densenet. [Электронный ресурс] URL: https://pytorch.org/hub/pytorch_vision_densenet/ (дата обращения: 26.02.2024 г.).

31. Новые архитектуры нейросетей. [Электронный ресурс] URL: https://habr.com/ru/articles/498168/ (дата обращения: 26.02.2024 г.).

32. Simonyan K., Zisserman A. Very deep convolutional networks for large-scale image recognition // arXiv.org, 2014. Дата обновления: 10.04.2015 г. URL: https://arxiv.org/abs/1409.1556 (дата обращения: 12.05.2024 г.).

33. VGG-Net Architecture Explained. [Электронный ресурс] URL: https://medium.com/@siddheshb008/vgg-net-architecture-explained-71179310050f (дата обращения: 26.02.2024 г.).

34. Zoph B., Vasudevan V., Shlens J., Quoc.V. Learning Transferable Architectures for Scalable Image Recognition // arXiv.org, 2017. Дата обновления: 11.04.2018 г. URL: https://arxiv.org/abs/1707.07012 (дата обращения: 12.05.2024 г.).

35. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez N., Kaiser L., Polosukhin I. Attention Is All You Need // arXiv.org, 2017. Дата обновления: 02.08.2023 г. URL: https://arxiv.org/abs/1706.03762 (дата обращения: 12.05.2024 г.).

36. Эмбеддинги для начинающих. [Электронный ресурс] URL: https://habr.com/ru/companies/otus/articles/787116/ (дата обращения: 10.04.2024 г.).

37. Illustrated Guide to Transformers- Step by Step Explanation. [Электронный ресурс] URL: https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0 (дата обращения: 10.04.2024 г.).

38. Dosovitskiy A., Beyer L., Kolesnikov A., Weissenborn D., Zhai X., Unterthiner T., Dehghani M., Minderer M., Heigold G., Gelly S., Uszkoreit J., Houlsby N. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale // arXiv.org, 2020. Дата обновления: 03.06.2021 г. URL: https://arxiv.org/abs/2010.11929 (дата обращения: 12.05.2024 г.).

39. Vision Transformer base-sized model. [Электронный ресурс] URL: https://huggingface.co/google/vit-base-patch16-224-in21k (дата обращения: 10.04.2024 г.).

40. CrossEntropyLoss. [Электронный ресурс] URL: https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html (дата обращения: 02.05.2024 г.).

41. Devlin J., Chang M., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding // arXiv.org, 2018. Дата обновления: 24.05.2019 г. URL: https://arxiv.org/abs/1810.04805 (дата обращения: 12.05.2024 г.)

42. Multimodal-AM. [Электронный ресурс] URL: https://www.kaggle.com/code/andreazecca/multimodal-am (дата обращения: 02.05.2024 г.).

43. MultiModalModel. [Электронный ресурс] URL: https://github.com/RGGU0/MultiModalModel (дата обращения: 02.05.2024 г.).

ПРИЛОЖЕНИЯ

Приложение А. Графики обучения с предобработкой и без

Графики обучения с применением и без применения предобработки набора данных приведены на рисунках 1–10.

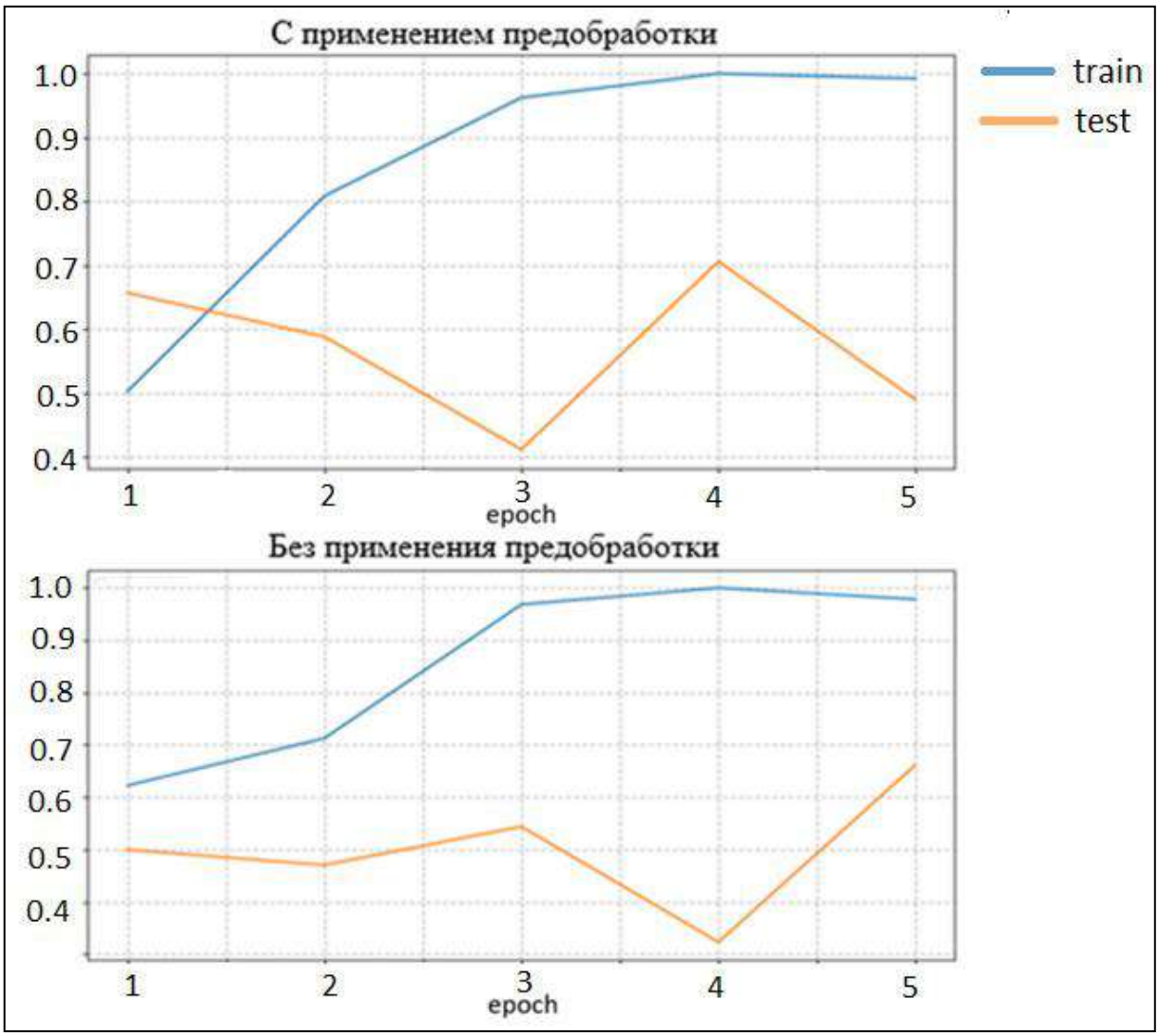

Рисунок 1 – Изменение accuracy при обучении DenseNet121

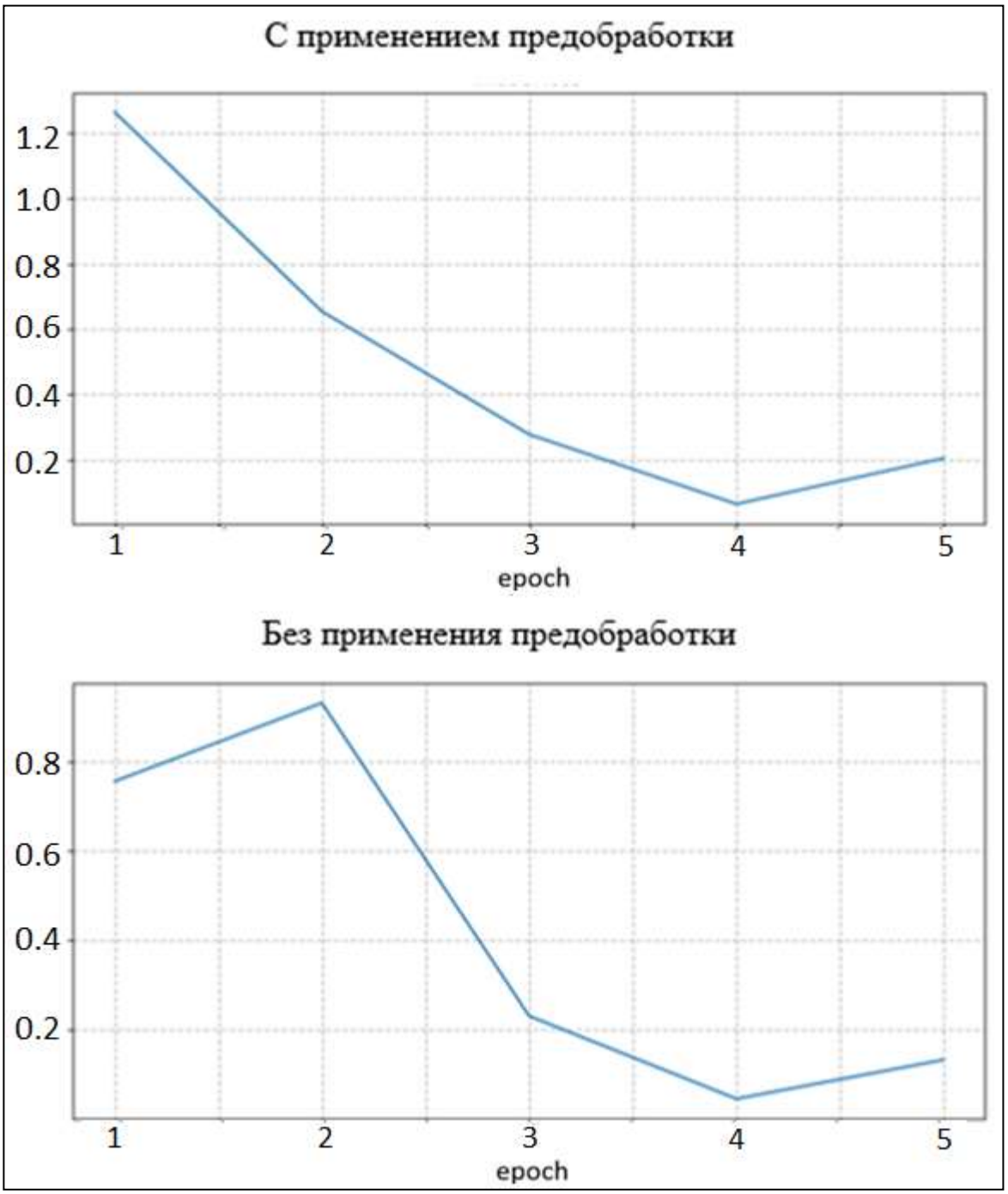

Рисунок 2 – Изменение loss при обучении DenseNet121

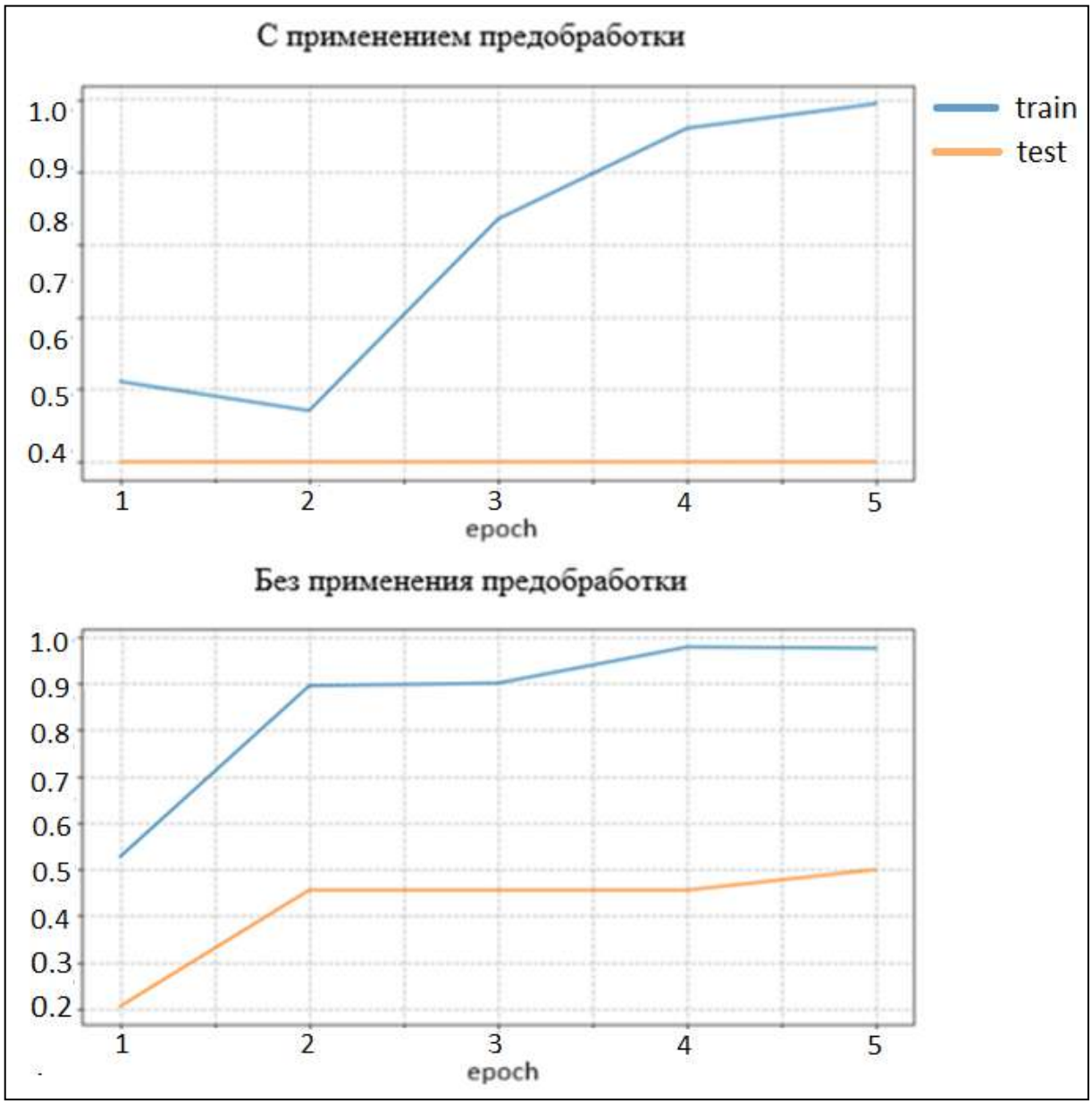

Рисунок 3 – Изменение accuracy при обучении DenseNet169

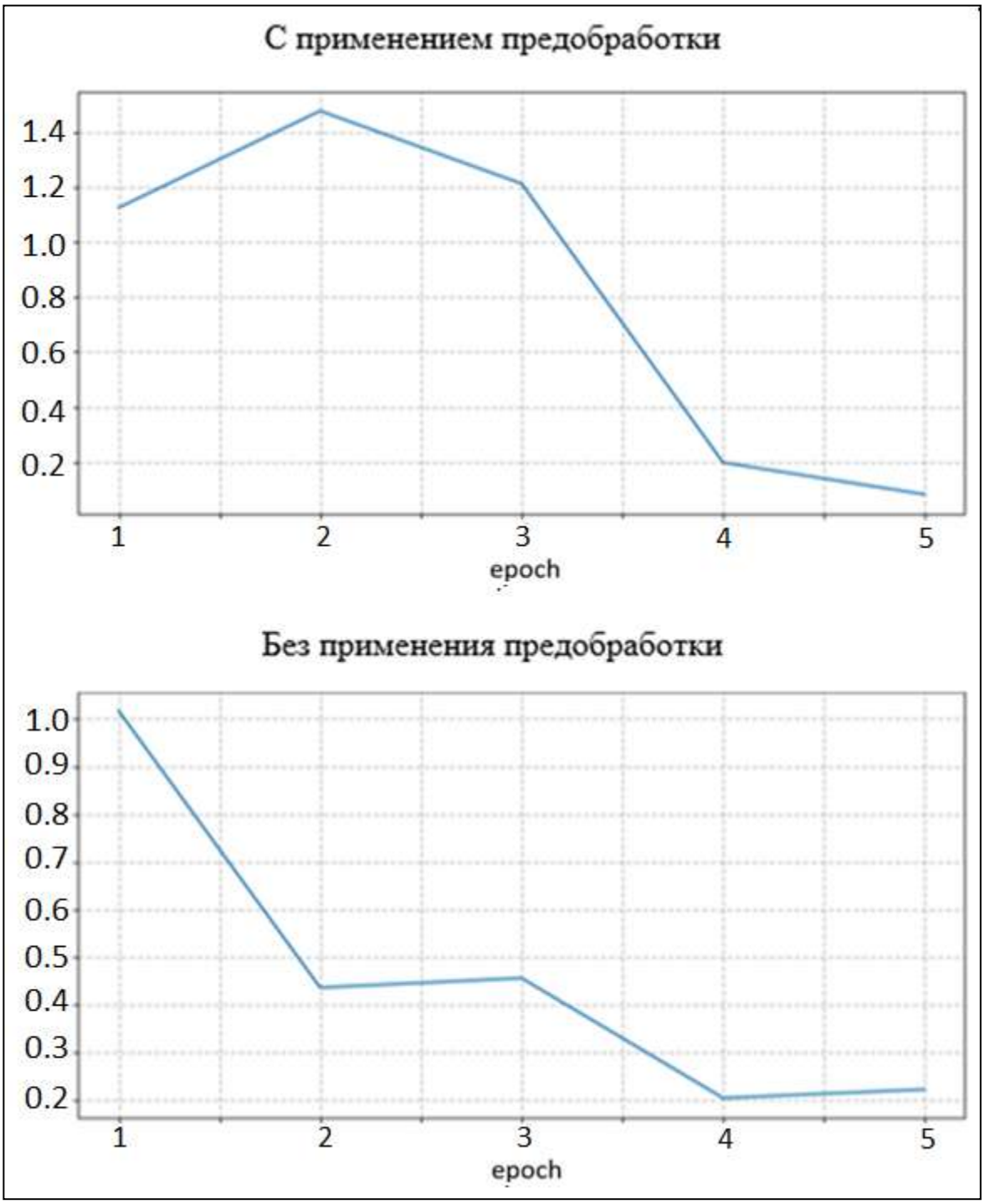

Рисунок 4 – Изменение loss при обучении DenseNet169

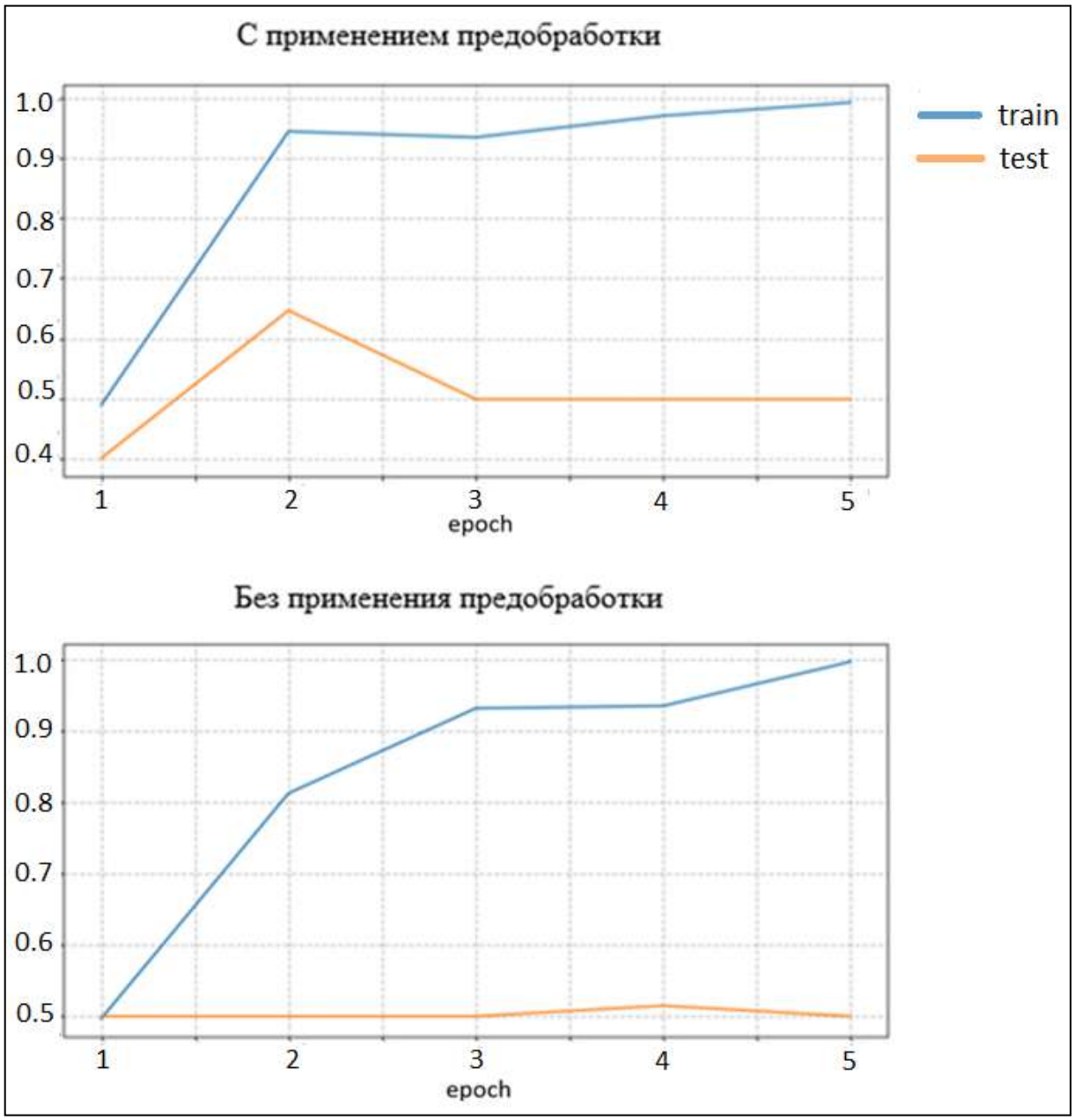

Рисунок 5 – Изменение accuracy при обучении DenseNet201

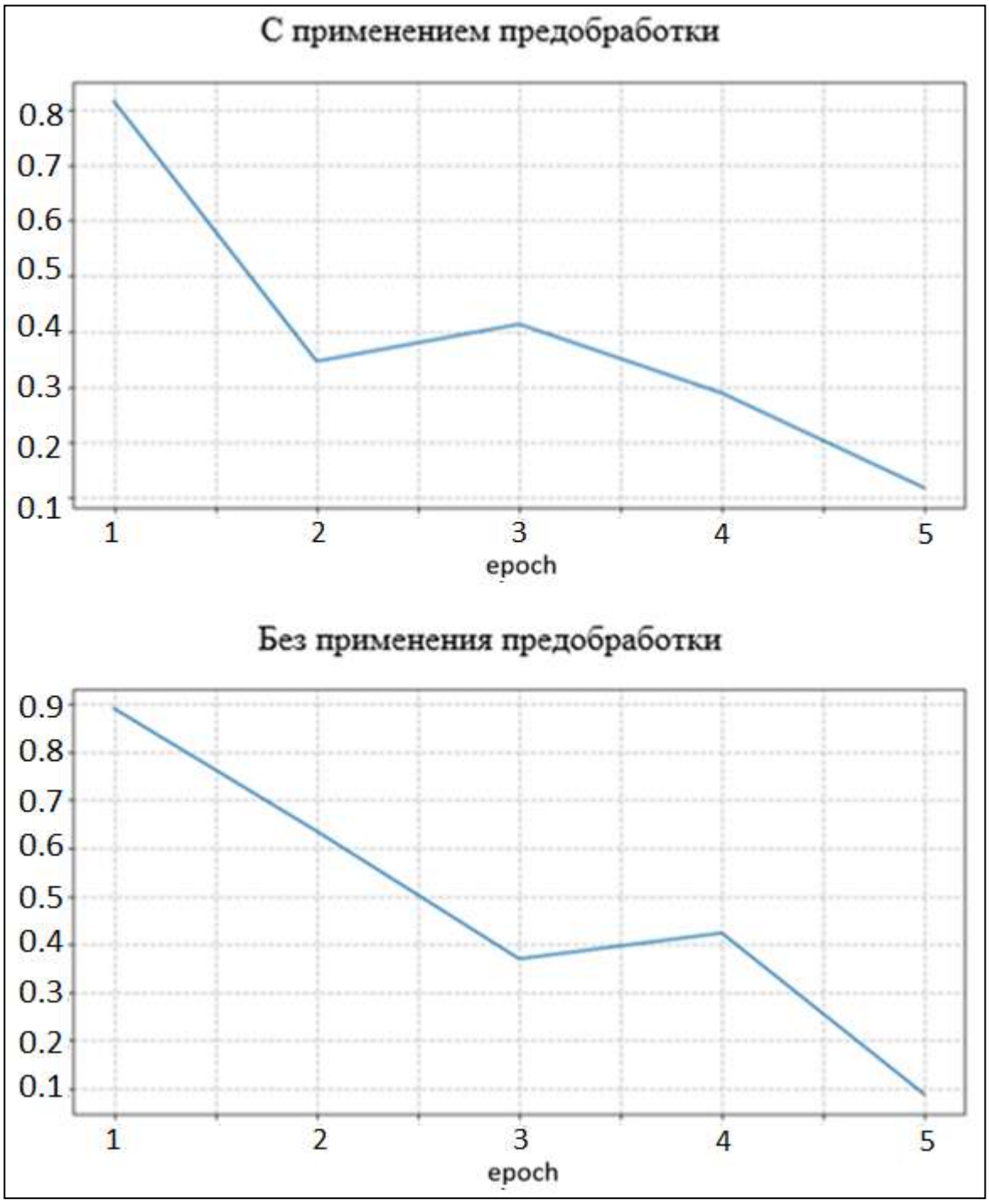

Рисунок 6 – Изменение loss при обучении DenseNet201

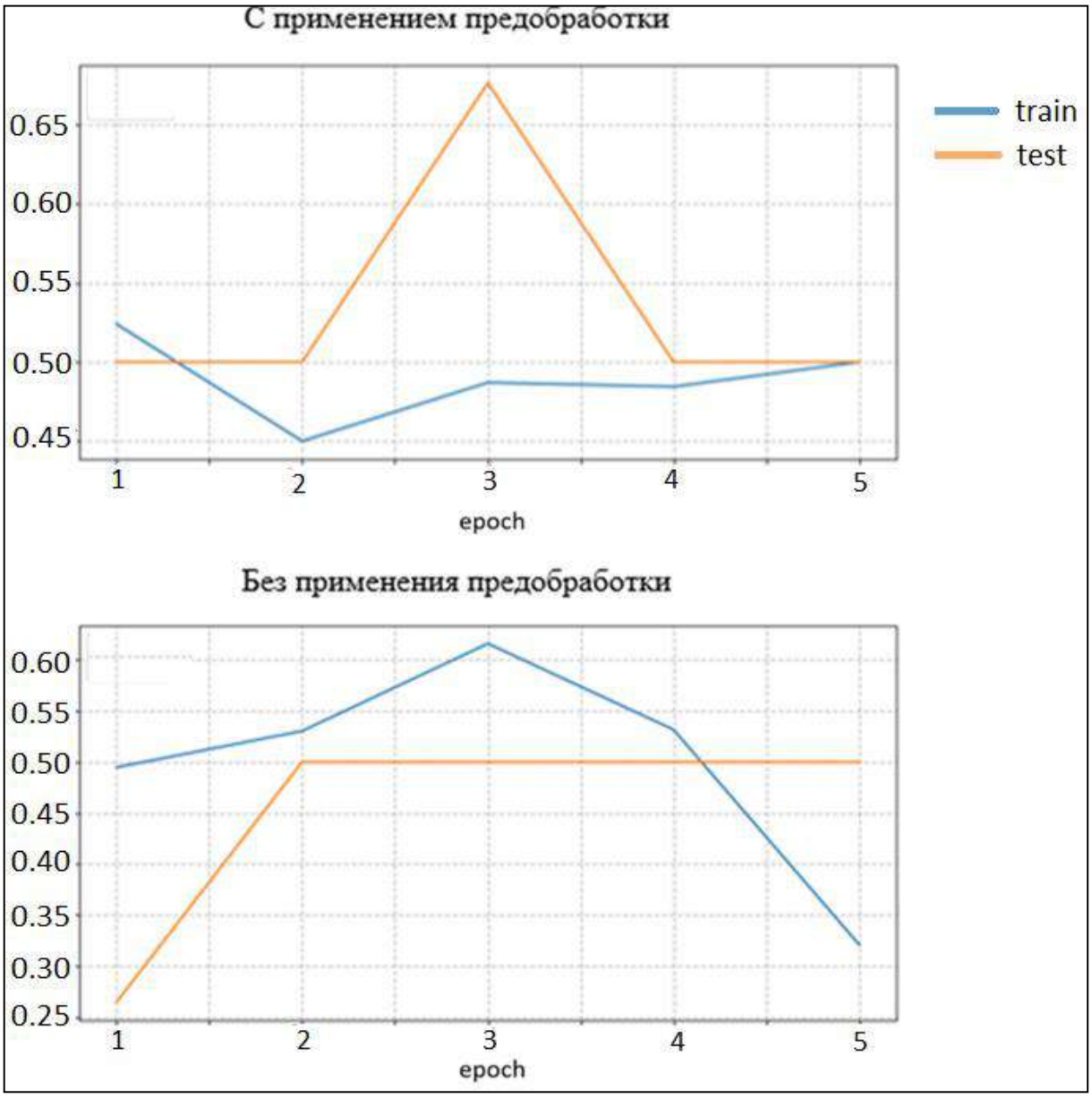

Рисунок 7 – Изменение accuracy при обучении VGG19

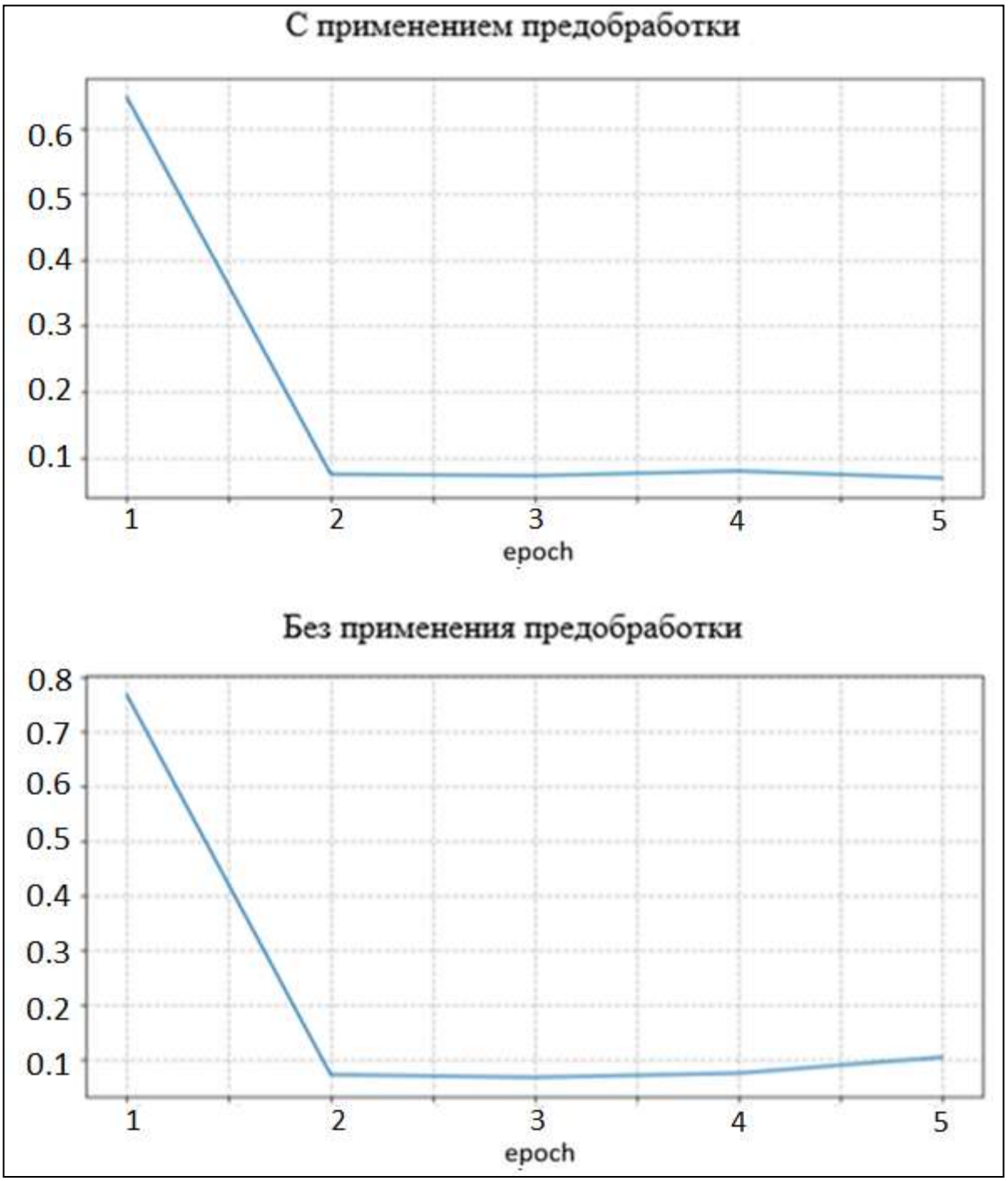

Рисунок 8 – Изменение accuracy при обучении VGG19

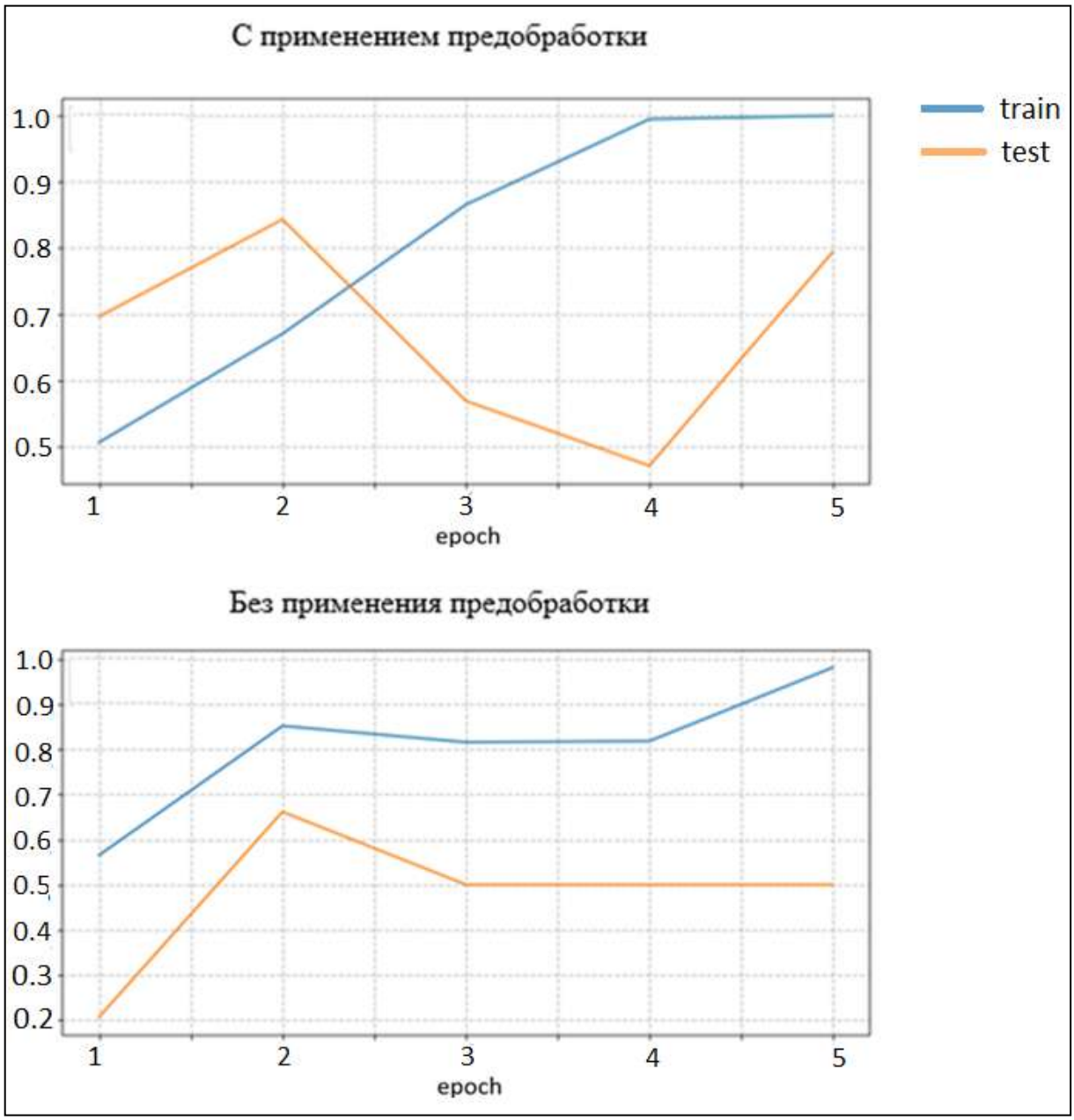

Рисунок 9 – Изменение accuracy при обучении NASNetLarge

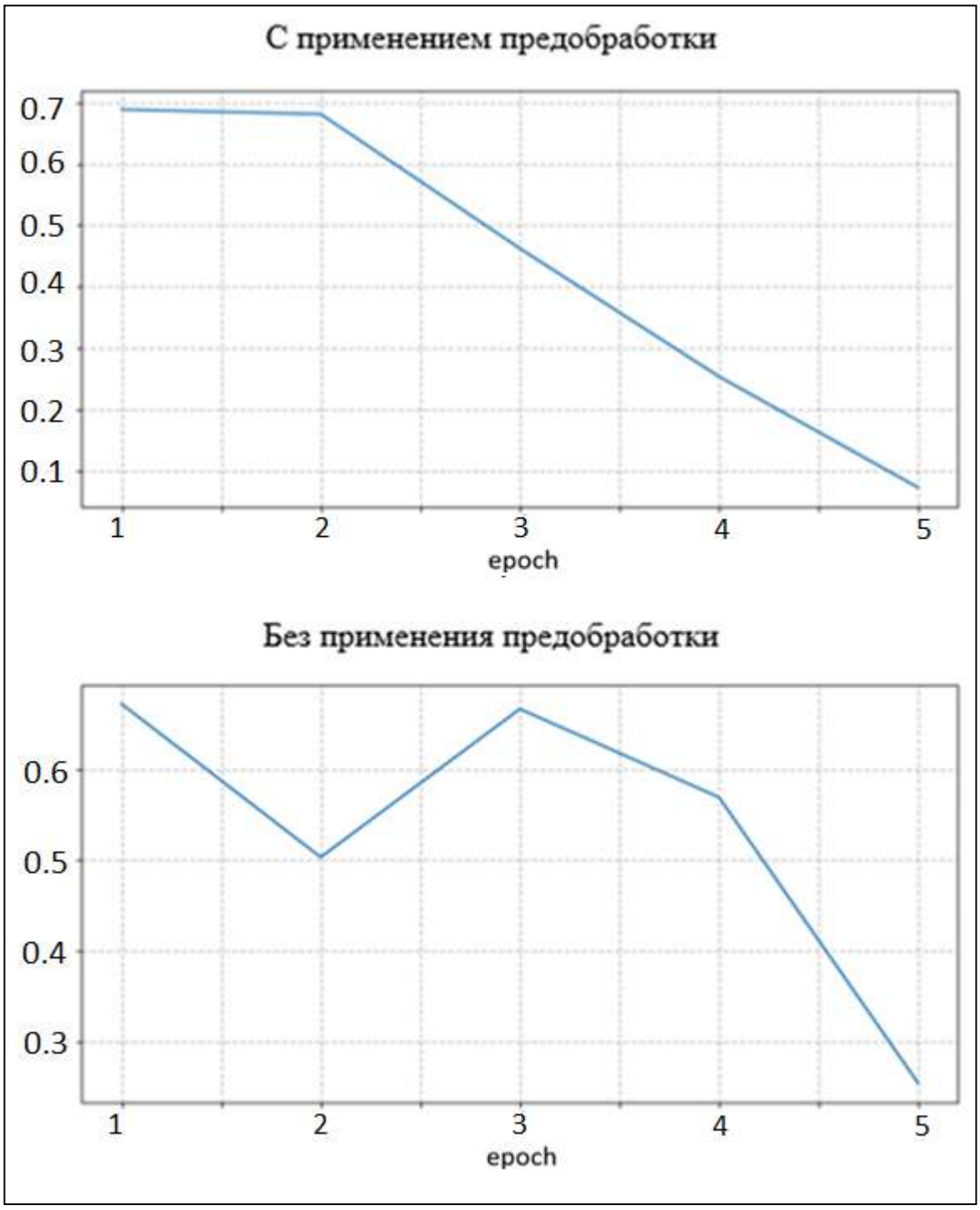

Рисунок 10 – Изменение loss при обучении NASNetLarge

Приложение Б. Листинги создания моделей

Листинг 1 – Функция создания модели для сегментации легких на снимках

```
def ModelEnhancer():
    input_shape = (500, 500, 3)
    VGG16_weight = "vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5"
    VGG16 = vgg16.VGG16(include_top=False, weights=VGG16_weight,
                        input_shape=input_shape)
    last_layer = VGG16.output
    set_trainable = False
    for layer in VGG16.layers:
        if layer.name in ['block1_conv1']:
            set_trainable = True
        if layer.name in ['block1_pool','block2_pool','block3_pool'
                          ,'block4_pool','block5_pool']:
            layer.trainable = False
    model_ = Conv2DTranspose(256,(3,3),strides=(2, 2))(last_layer)
    model_ = LeakyReLU(0.1)(model_)
    model_ = BatchNormalization()(model_)
    concat_1 = concatenate([model_,VGG16.get_layer("block5_conv3").output])
    model_ = Conv2D(512,(3,3),strides=(1, 1),padding='same')(concat_1)
    model_ = LeakyReLU(0.1)(model_)
    model_ = BatchNormalization()(model_)
    model_ = Conv2DTranspose(512,(3,3),
                        strides=(2, 2),padding='same')(model_)
    model_ = LeakyReLU(0.1)(model_)
    model_ = BatchNormalization()(model_)
    concat_2 = concatenate([model_,VGG16.get_layer("block4_conv3").output])
    model_ = Conv2D(512,(3,3),strides=(1, 1),padding='same')(concat_2)
    model_ = LeakyReLU(0.1)(model_)
    model_ = BatchNormalization()(model_)
    model_ = Conv2DTranspose(512,(3,3),strides=(2, 2))(model_)
    model_ = LeakyReLU(0.1)(model_)
    model_ = BatchNormalization()(model_)
    concat_3 = concatenate([model_,VGG16.get_layer("block3_conv3").output])
    model_ = Conv2D(256,(3,3),strides=(1, 1),padding='same')(concat_3)
    model_ = LeakyReLU(0.1)(model_)
    model_ = BatchNormalization()(model_)
    model_ = Conv2DTranspose(256,(3,3),
                        strides=(2, 2),padding='same')(model_)
    model_ = LeakyReLU(0.1)(model_)
    model_ = BatchNormalization()(model_)
    concat_4 = concatenate([model_,VGG16.get_layer("block2_conv2").output])
    model_ = Conv2D(128,(3,3),strides=(1, 1),padding='same')(concat_4)
    model_ = LeakyReLU(0.1)(model_)
    model_ = BatchNormalization()(model_)
    model_ = Conv2DTranspose(128,(3,3),
                        strides=(2, 2),padding='same')(model_)
    model_ = LeakyReLU(0.1)(model_)
    model_ = BatchNormalization()(model_)
    concat_5 = concatenate([model_,VGG16.get_layer("block1_conv2").output])
    model_ = Conv2D(64,(3,3),strides=(1, 1),padding='same')(concat_5)
    model_ = LeakyReLU(0.1)(model_)
    model_ = BatchNormalization()(model_)
    model_ = Conv2D(3,(3,3),strides=(1, 1),padding='same')(model_)
    model_ = LeakyReLU(0.1)(model_)
    model_ = BatchNormalization()(model_)
    model_ = Model(VGG16.input,model_)
    return model_
```

Листинг 2 – Класс мультимодальной модели

```
class MulTA(nn.Module):
    def __init__(self, embedding_dim, d_ffn, n_blocks, head, labels=0,
        dropout_prob=0.1,hidden_state_index=8):
        super().__init__()
        self.Linear = nn.Linear(465708, 197)
        self.vit = model_preterened_vit
        self.embedding_dim = embedding_dim
        self.d_ffn = d_ffn
        self.n_blocks = n_blocks
        self.head = head
        self.dropout_prob = dropout_prob
        self.text_crossmodal_blocks = nn.ModuleList([
            MulTA_CrossAttentionBlock(
                self.embedding_dim, self.d_ffn,
                dropout_prob=self.dropout_prob
                ) for _ in range(self.n_blocks)])
        self.img_crossmodal_blocks = nn.ModuleList([
            MulTA_CrossAttentionBlock(
                self.embedding_dim, self.d_ffn,
                dropout_prob=self.dropout_prob
                ) for _ in range(self.n_blocks)])
        self.pos_encoder = PositionalEncoding(embedding_dim,
                                              dual_modality=False)
        self.hidden_state_index = hidden_state_index
    def forward(self, text, pixel_values, labels=0,
                        decoder_start_token_id=0):
        embedder_output = embedder(text['input_ids'],
                        output_hidden_states=True)
        if self.hidden_state_index == -1:
          text_features = embedder_output['last_hidden_state']
        else:
    text_features=embedder_output['hidden_states'][self.hidden_state_index]
        text_features = self.pos_encoder(text_features)
        text_attentions = text['attention_mask']
        img_features = pixel_values
        img_features = torch.tensor(img_features)
        img_features = img_features.unsqueeze(0)
        outputs_att = self.vit(img_features, output_attentions=True,
                               interpolate_pos_encoding=True,
                               output_hidden_states = True)
        img_attentions = outputs_att.attentions
        img_attentions = self.Linear(
                        torch.tensor(img_attentions[0]).flatten(1))
        img_features = outputs_att.hidden_states
        text_crossmodal_out = text_features
        for cm_block in self.text_crossmodal_blocks:
            text_crossmodal_out = cm_block(text_crossmodal_out,
                                           img_features[0], img_attentions)
        img_crossmodal_out = img_features[0]
        for cm_block in self.img_crossmodal_blocks:
            img_crossmodal_out = cm_block(img_crossmodal_out,
                                          text_features, text_attentions)
        text_crossmodal_out_mean = torch.mean(text_crossmodal_out, dim=1)
        img_crossmodal_out_mean = torch.mean(img_crossmodal_out, dim=1)
        text_img = torch.cat((text_crossmodal_out_mean,
                              img_crossmodal_out_mean), dim=-1)
        head_out = self.head(text_img)
        return head_out
```