Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет	*Кафедра*
вычислительной техники	«Вычислительная техника»

Направление подготовки: 09.04.01 «Информатика и вычислительная техника»

Магистерская программа: «Прикладной искусственный интеллект»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему

ПРИМЕНЕНИЕ СВЕРТОЧНЫХ НЕЙРОННЫХ СЕТЕЙ ДЛЯ АВТОМАТИЧЕСКОЙ КЛАССИФИКАЦИИ МЕДИЦИНСКИХ СНИМКОВ

Студент 15.06.2026 (подпись, дата) Шадрин Данила Алексеевич (ФИО полностью)

Руководитель 15.06.2026 (подпись, дата) С.А. Зинкин (фамилия, инициалы)

Нормоконтролёр 15.06.2026 (подпись, дата) А.С. Бычков (фамилия, инициалы)

Рецензент доцент ПензГТУ (должность, место работы) 16.06.26 (подпись, дата) С.В. Печерский (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 22.05.2026 № 14)

Заведующий кафедрой (подпись) М.А. Митрохин (фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от 23.06.2026 №29)

Секретарь ГЭК (подпись) Е.И. Гурин (фамилия, инициалы)

Пенза 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет	*Кафедра*
вычислительной техники	«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
_____М.А. Митрохин
«20» 09 2024 г.

ЗАДАНИЕ
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ

1. Студент Шадрин Данила Алексеевич гр. 24ВВИм1 факультета ВТ

2.Тема работы Применение сверточных нейронных сетей для автоматической классификации медицинских снимков

Тема утверждена приказом ПГУ № 920/о от "20" 09 2024 г.

3. Руководитель магистерской диссертации д.т.н., профессор кафедры ВТ ПГУ Зинкин С.А.

4. Задание на работу (назначение разработки, исходные данные и т.п.)
Выполнить исследование и разработать программный комплекс для автоматической классификации медицинских снимков с использованием сверточной нейронной сети. Создать web-приложение, позволяющее загружать обезличенные рентгеновские снимки, выполнять предварительную обработку изображений, запускать классификацию на основе ResNet50, формировать визуальное пояснение результата средствами Grad-CAM и сохранять отчет по результатам анализа. Программный комплекс должен включать клиентскую часть, API-сервис, модуль подготовки данных, нейросетевой модуль и подсистему отчетов.

5. Перечень подлежащих разработке вопросов
Требуется выполнить анализ предметной области автоматической классификации медицинских снимков, рассмотреть особенности рентгеновских изображений органов грудной клетки и датасета ChestX-ray14. Необходимо исследовать применение сверточных нейронных сетей и архитектуры ResNet50, описать подход переноса обучения, метрики качества многометочной классификации и метод визуального пояснения Grad-CAM. Требуется разработать архитектуру приложения, включающую web-интерфейс, REST API, модуль предварительной обработки PNG, JPG и DICOM-изображений, сервис классификации и подсистему хранения результатов.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	10%	20.09.24-15.11.2024	
2.	Анализ требований к программному комплексу	10%	16.11.2024–20.12.2024	
3.	Изучение литературы и существующих решений	10%	21.12.2024–31.01.2025	
4.	Проектирование архитектуры web-приложения	15%	01.02.2025–15.03.2025	
5.	Проектирование программного комплекса	30%	16.03.2025–31.01.2026	
6.	Тестирование разработанной системы	10%	01.02.2026–28.02.2026	
7.	Анализ полученных результатов исследования и выбор наиболее подходящих моделей	10%	01.03.2026–30.04.2026	
8.	Оформление пояснительной записки	5%	01.05.2026–20.05.2026	

Дата выдачи задания "20" 09 2024 г.

Руководитель магистерской диссертации _____ Зинкин С.А.
(подпись, дата)

Задание к исполнению принял студент _____ Шадрин Д.А.
(подпись, дата)

Работу к защите допустить
Декан факультета 17.06.2026 Фионова Л.Р.

РЕФЕРАТ

Пояснительная записка содержит 105 листов, 9 рисунков, 18 таблиц, 11 источников, 2 приложения.

СВЕРТОЧНАЯ НЕЙРОННАЯ СЕТЬ, CNN, RESNET50, МЕДИЦИНСКИЕ СНИМКИ, РЕНТГЕНОГРАММА, CHESTX-RAY14, МНОГОМЕТОЧНАЯ КЛАССИФИКАЦИЯ, GRAD-CAM, FASTAPI, PYTORCH, REACT

Целью работы является разработка программного комплекса для автоматической классификации медицинских снимков с использованием сверточной нейронной сети. Приложение предназначено для загрузки обезличенных рентгеновских снимков грудной клетки, их предварительной обработки, классификации по рентгенологическим признакам и формирования отчета.

В работе рассмотрены особенности медицинских изображений, набор данных ChestX-ray14, архитектура ResNet50, перенос обучения, метрики оценки качества и метод визуального пояснения Grad-CAM. Для классификации используется модель ResNet50 с четырнадцатью выходами, соответствующими классам ChestX-ray14.

В результате реализована web-система MedVision CNN. Серверная часть построена на FastAPI и PyTorch, клиентская часть - на React. Система поддерживает загрузку PNG, JPG и DICOM-изображений, выполняет предварительную обработку снимков, рассчитывает вероятности по классам, строит Grad-CAM-карту и сохраняет отчет.

Результат работы системы имеет исследовательское назначение и не является медицинским диагнозом.

					ПГУ 1.090401.28.001 ПЗ		
Изм.	Лист	№ докум	Подпись	Дата			
Разраб.		Шадрин Д.А.			*Применение сверточных нейронных сетей для автоматической классификации медицинских снимков Пояснительная записка*	Лит.	Лист 2 / Листов 105
Провер.		Зинкин С.А.					
Н. Контр.		Бычков А.С.				ФВТ, каф. ВТ гр. 24ВВИм1	
Утв.		Митрохин М.А.					

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ 12
ВВЕДЕНИЕ 13
1 Анализ предметной области 15
1.1 Медицинские снимки как объект автоматического анализа 16
1.2 Постановка задачи автоматической классификации 18
1.3 Требования к разрабатываемой системе 18
1.4 Набор данных ChestX-ray14 21
1.5 Разделение данных и качество разметки 21
1.6 Предварительная обработка изображений 24
1.7 Выбор архитектуры ResNet50 24
1.8 Метрики оценки качества 26
1.9 Визуальное пояснение результата 27
1.10 Ограничения применения автоматической классификации 27
1.11 Вывод по разделу 28
2 Архитектура сверточных нейронных сетей 30
2.1 Представление изображения 30
2.2 Сверточный слой 31
2.3 Функции активации, подвыборка и нормализация 31
2.4 Остаточные блоки ResNet50 32
2.5 Многометочный выход модели 33
2.6 Перенос обучения 33
2.7 Оценка качества CNN 34
2.8 Grad-CAM в структуре CNN 34
2.9 Вывод по разделу 35
3 Анализ аналогов 36
3.1 Критерии сравнения 36
3.2 CheXNet 36
3.3 TorchXRayVision 37
3.4 MONAI 38

3.5 Коммерческие системы анализа снимков 38
4 Архитектура приложения 41
4.1 Общая схема 41
4.2 Клиентская часть 42
4.3 Серверная часть и API 43
4.4 Модуль предварительной обработки 44
4.5 Сервис классификации 45
4.6 Модуль Grad-CAM 45
4.7 Хранение отчетов 46
4.8 Обработка ошибок 47
5 Реализация приложения 48
5.1 Подготовка датасета ChestX-ray14 48
5.2 Реализация загрузки и предварительной обработки снимков 51
5.3 Реализация модели ResNet50 52
5.4 Обучение модели 54
5.5 Оценка качества модели 56
5.6 Анализ ошибок 59
5.7 Реализация Backend API 59
5.8 Хранение отчетов 63
5.9 Реализация Grad-CAM в приложении 64
5.10 Реализация клиентской части 65
5.11 Запуск приложения 67
6 Тестирование и проверка работы системы 68
6.1 Проверка подготовки данных 68
6.2 Проверка обучения и оценки 69
6.3 Проверка API и обработки ошибок 70
6.4 Проверка пользовательского интерфейса 72
6.5 Анализ ограничений 73
6.6 Вывод по тестированию 74
7 Итоговый результат разработки 76
ЗАКЛЮЧЕНИЕ 77

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 78
ПРИЛОЖЕНИЕ А – Листинг программы 80
ПРИЛОЖЕНИЕ Б – Презентация 103

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей работе применяются следующие сокращения и обозначения.

AI (Artificial Intelligence) - искусственный интеллект

API (Application Programming Interface) - программный интерфейс приложения

AUROC (Area Under Receiver Operating Characteristic Curve) - площадь под ROC-кривой

BCE (Binary Cross-Entropy) - бинарная кросс-энтропия

CNN (Convolutional Neural Network) - сверточная нейронная сеть

DICOM (Digital Imaging and Communications in Medicine) - стандарт хранения и передачи медицинских изображений

F1-score - метрика, отражающая баланс между точностью и полнотой

FastAPI - web-фреймворк Python для разработки REST API

Grad-CAM (Gradient-weighted Class Activation Mapping) - метод визуального пояснения решений сверточной сети

GPU (Graphics Processing Unit) - графический процессор

JSON (JavaScript Object Notation) - текстовый формат обмена данными

PNG, JPG, JPEG - распространенные форматы изображений

PyTorch - библиотека машинного обучения, используемая для реализации и обучения нейронной сети

React - библиотека для разработки клиентской части web-приложения

REST (Representational State Transfer) - архитектурный стиль взаимодействия web-сервисов

ResNet50 - архитектура сверточной нейронной сети с остаточными блоками;

SQLite - встраиваемая реляционная база данных

Threshold - порог принятия решения для класса

TP, FP, FN, TN - истинно положительные, ложноположительные, ложноотрицательные и истинно отрицательные результаты

ВВЕДЕНИЕ

Методы искусственного интеллекта все чаще используют для анализа изображений. Особенно заметно это в медицинской визуализации. Рентгенография, компьютерная томография и магнитно-резонансная томография дают большой поток визуальных данных, которые врач должен внимательно просматривать. Часть такой работы можно автоматизировать с помощью сверточных нейронных сетей.

В работе рассматривается задача автоматической классификации рентгеновских снимков органов грудной клетки. Такой тип изображений выбран из-за практической значимости и наличия открытых исследовательских наборов данных. Рентгенограммы грудной клетки применяются при выявлении различных патологических признаков, а один снимок может содержать сразу несколько наблюдений. Поэтому задача рассматривается как многометочная классификация.

За основу программного решения взята сверточная нейронная сеть ResNet50. Эта архитектура используется вместе с переносом обучения, что дает возможность применять ранее обученные признаки изображения и затем адаптировать модель к медицинскому набору данных. Для обучения и оценки качества используется ChestX-ray14 - открытый набор рентгенограмм грудной клетки с разметкой по четырнадцати классам [1].

Отдельно рассматривается объяснимость результата. В медицинских задачах одной числовой вероятности класса недостаточно. Нужно видеть, какие области снимка повлияли на решение модели. Для этого используется Grad-CAM: метод строит тепловую карту областей, связанных с выбранным выходом сверточной сети [2].

Цель работы - разработка программного комплекса для автоматической классификации медицинских снимков с использованием сверточной нейронной сети. Для этого нужно выполнить анализ предметной области, рассмотреть архитектуру CNN и ResNet50, проанализировать существующие

решения, а также спроектировать архитектуру приложения для загрузки, обработки и классификации снимков.

Результаты работы имеют исследовательское назначение. Автоматический вывод модели не считается медицинским диагнозом и может рассматриваться только как результат работы программного прототипа. Практическое применение подобной системы требует отдельной клинической валидации и экспертной оценки.

1 Анализ предметной области

Разработка программного комплекса для автоматической классификации медицинских снимков требует предварительного анализа самой предметной области, характера входных данных и ограничений, которые возникают при использовании методов искусственного интеллекта в медицине. В отличие от задач с обычными изображениями, здесь важны не только визуальные признаки, но и корректность подготовки данных, воспроизводимость результата, возможность объяснить поведение модели и аккуратное отношение к персональным медицинским сведениям.

В работе рассматривается задача анализа рентгеновских снимков органов грудной клетки. Этот тип исследований широко применяется в клинической практике и используется для выявления признаков различных состояний: увеличения сердца, воспалительных изменений, скопления жидкости, участков уплотнения, пневмоторакса и других наблюдений. Для программной системы такая постановка является сложной, поскольку один снимок может содержать несколько признаков одновременно, а отдельные признаки могут быть выражены слабо.

Классические методы компьютерного зрения в подобных задачах требуют ручного описания признаков. Разработчику нужно заранее определить, какие участки изображения следует анализировать и какие числовые характеристики извлекать. В медицинских снимках такой подход ограничен: визуальные проявления могут отличаться в зависимости от оборудования, положения пациента, качества снимка и сопутствующих изменений. Поэтому в работе выбран подход на основе сверточных нейронных сетей, которые позволяют автоматически выделять признаки в процессе обучения [3].

Общая логика обработки снимка в системе показана на рисунке 1.

Рисунок 1 – Общая схема обработки медицинского снимка

Система должна принимать входной снимок, выполнять его предварительную обработку, передавать изображение в нейросетевую модель, формировать вероятности по целевым классам и дополнять результат визуальным пояснением. Такой порядок выбран, чтобы отделить подготовку данных, классификацию и показ результата пользователю.

1.1 Медицинские снимки как объект автоматического анализа

Медицинское изображение отличается от обычной фотографии прежде всего назначением. Фотография чаще всего описывает видимый объект в привычных для человека условиях, а медицинский снимок отражает внутренние структуры организма или результат взаимодействия тканей с излучением. В случае рентгенографии изображение строится по различию в поглощении рентгеновского излучения тканями. Плотные структуры, например кости, выглядят светлее, а области, содержащие воздух, обычно темнее.

При анализе рентгенограммы грудной клетки врач оценивает не отдельный участок, а всю совокупность признаков: форму легочных полей, контуры сердца, положение диафрагмы, наличие затемнений, изменения прозрачности, смещение средостения и другие элементы. Для автоматической

модели это означает необходимость учитывать как локальные признаки, так и общую структуру снимка.

Сложность автоматической обработки связана еще и с тем, что изображения могут заметно различаться между собой. Разные аппараты, параметры съемки, положение пациента и качество цифровой обработки влияют на яркость, контрастность и уровень шума. На снимках могут присутствовать технические маркеры, подписи, рамки и другие элементы, которые не имеют диагностического значения, но могут повлиять на модель, если не уделить внимание подготовке данных.

Дополнительная трудность состоит в том, что рентгенограмма не всегда содержит явно выраженный объект, который можно отделить от фона. На изображении одновременно присутствуют анатомические структуры, технические элементы и последствия положения пациента во время исследования. Поэтому алгоритм должен работать не с отдельным фрагментом, а с изображением в целом, сохраняя при этом способность выделять локальные изменения.

При разработке системы учитывается, что снимки могут иметь разную область кадрирования. На одних изображениях полностью видна грудная клетка, на других часть области может быть обрезана. Это влияет на предварительную обработку: сильное обрезание или некорректное масштабирование способно удалить полезные признаки. Поэтому выбран нейтральный вариант подготовки, при котором изображение приводится к входному размеру модели без агрессивного удаления областей.

Для рассматриваемой задачи важно, чтобы программный комплекс не только выдавал результат классификации, но и сохранял возможность контроля. Система должна показывать, на какие области изображения ориентировалась модель, и не должна скрывать ограничения автоматического анализа. Поэтому в состав решения включается механизм визуального пояснения результата.

1.2 Постановка задачи автоматической классификации

Автоматическая классификация медицинского снимка заключается в определении набора признаков, которые могут присутствовать на изображении. В простейшем случае классификация сводится к выбору одного класса из нескольких. Однако для рентгенограмм грудной клетки такая постановка не всегда корректна, поскольку на одном изображении могут одновременно наблюдаться разные изменения.

В работе используется многометочная постановка. Модель получает изображение и формирует набор независимых оценок для каждого класса. Например, для одного снимка могут быть одновременно повышены вероятности классов Effusion и Cardiomegaly. Это отличается от многоклассовой классификации, где предполагается выбор только одного класса.

Формально входное изображение можно обозначить как X, а выход модели как набор вероятностей (1):

$$p = (p_1, p_2, \dots, p_k), \text{где } p_i = \sigma(z_i) \quad (1)$$

где K – число целевых классов;

z_i – выход модели до применения функции активации;

p_i – вероятность наличия i-го признака;

σ – сигмоидальная функция.

Сигмоидальная функция используется потому, что каждый класс оценивается независимо от остальных (2):

$$\sigma(\mathrm{z}) = \frac{1}{1+e^{-z}} \quad (2)$$

После получения вероятностей применяется порог принятия решения. Если вероятность класса выше заданного порога, соответствующий признак

включается в результат анализа. В базовом варианте порог может быть равен 0,5, однако при последующей настройке модели пороги лучше подбирать отдельно для каждого класса, поскольку распространенность и клиническая значимость признаков различаются.

Такой подход дает возможность одной моделью оценивать несколько рентгенологических наблюдений. Результат модели при этом нельзя трактовать как диагноз; это исследовательская оценка, которую должен проверить специалист.

Многометочная постановка также влияет на интерфейс приложения. Пользователю показывается не один итоговый класс, а полный список вероятностей. Это помогает увидеть, что модель могла дать несколько близких оценок, даже если порог прошел только один класс. Такая информация полезна при анализе поведения модели и при выборе порогов для отдельных наблюдений.

Важной частью постановки является выбор порога. Один общий порог удобен для первого эксперимента, но не всегда дает сбалансированный результат. Классы в ChestX-ray14 встречаются с разной частотой, поэтому в программе предусмотрено хранение индивидуальных порогов. Это делает вывод более гибким и дает возможность корректировать поведение модели без изменения самой архитектуры.

1.3 Требования к разрабатываемой системе

Перед выбором модели и инструментов разработки нужно определить требования к программному комплексу. Они делятся на функциональные, технологические и требования, связанные с особенностями медицинских данных.

Функционально система должна обеспечивать полный путь работы с изображением: загрузку снимка, проверку формата, предварительную обработку, запуск модели, формирование результата и сохранение отчета. Пользователь не должен выполнять дополнительные действия, связанные с подготовкой изображения вручную.

Основные требования к разрабатываемой системе приведены в таблице 1.

Таблица 1 – Требования к программному комплексу

Группа требований	Содержание требования	Назначение
Входные данные	Поддержка PNG, JPG и DICOM	Возможность работы с пользовательскими снимками и исследовательскими данными
Предварительная обработка	Нормализация, изменение размера, приведение к формату модели	Получение воспроизводимого входа для нейронной сети
Классификация	Многометочный выход по 14 классам ChestX-ray14	Учет нескольких возможных признаков на одном снимке
Объяснимость	Формирование Grad-CAM карты	Проверка областей изображения, повлиявших на вывод модели
Хранение результата	Сохранение вероятностей, версии модели и отчета	Воспроизводимость проведенного анализа
Безопасность данных	Отказ от сохранения персональных DICOM-метаданных	Снижение риска обработки лишних медицинских сведений

Отдельное требование связано с тем, как формулируется результат. Программный комплекс не должен создавать впечатление, что он сам ставит медицинский диагноз. Вывод модели сопровождается предупреждением об исследовательском назначении системы. Для учебной и исследовательской разработки это особенно важно, поскольку клиническая валидация еще не проводилась.

С технологической стороны система должна быть расширяемой. В текущей версии системы используется ResNet50, но архитектура приложения должна позволять в дальнейшем заменить модель или подключить другую версию весов без полной переработки интерфейса и API. По этой причине лучше разделить модули подготовки данных, инференса, объяснения результата и хранения отчетов.

К нефункциональным требованиям отнесена воспроизводимость. Пользовательский отчет должен содержать не только вероятности классов, но и версию модели, которая сформировала результат. Без этой информации невозможно понять, была ли классификация выполнена старой или новой версией весов после повторного обучения.

Ошибки тоже нужно обрабатывать понятно. Система сообщает пользователю, если файл не поддерживается, поврежден или превышает допустимый размер. Так приложение легче проверять, а внутренняя ошибка backend не попадает в интерфейс без пояснения.

API-сервис также отделяет внутренние Python-модули от клиентской части. Frontend не знает, как именно загружается модель или строится Grad-CAM; он получает только готовый JSON-ответ. Такое разделение снижает связность между слоями приложения и упрощает дальнейшую замену отдельных модулей.

Для маршрутов отчетов предусмотрено отдельное получение данных и изображений. Это дает возможность открыть сохраненный отчет позже, не выполняя повторную классификацию. Такой режим важен для проверки воспроизводимости и для демонстрации уже полученных результатов.

1.4 Набор данных ChestX-ray14

Для обучения и проверки модели выбран набор данных ChestX-ray14. Он содержит фронтальные рентгенограммы грудной клетки и разметку по четырнадцати рентгенологическим наблюдениям. Набор опубликован как открытый исследовательский ресурс и часто используется при проверке методов автоматической классификации рентгеновских снимков.

Главная особенность ChestX-ray14 состоит в многометочной разметке. Один снимок может относиться сразу к нескольким классам. Например, на изображении могут быть одновременно отмечены Effusion и Cardiomegaly. Поэтому данные нельзя рассматривать как обычную задачу выбора одного класса. Для каждого признака нужна отдельная оценка вероятности.

Классы, используемые в обучении модели, приведены в таблице 2. Метка No Finding в перечень патологических классов не включается. В программной части она используется как отсутствие положительных меток по всем выбранным наблюдениям.

Таблица 2 – Классы набора ChestX-ray14

Класс	Смысл метки в задаче
Atelectasis	Признаки ателектаза
Cardiomegaly	Увеличение размеров сердца
Effusion	Плевральный выпот
Infiltration	Инфильтративные изменения
Mass	Объемное образование
Nodule	Узелковое образование
Pneumonia	Признаки пневмонии
Pneumothorax	Пневмоторакс
Consolidation	Участок уплотнения легочной ткани
Edema	Признаки отека
Emphysema	Эмфизематозные изменения
Fibrosis	Фиброзные изменения
Pleural Thickening	Утолщение плевры
Hernia	Грыжевое выпячивание

В наборе данных классы распределены неравномерно. Одни наблюдения встречаются часто, другие представлены небольшим количеством примеров. Это нужно учитывать при обучении и оценке качества модели, поскольку высокая общая точность при дисбалансе классов может скрывать слабую работу по редким признакам.

Дисбаланс классов влияет не только на обучение, но и на последующую интерпретацию результатов. Часто встречающиеся классы дают модели больше примеров, поэтому сеть быстрее находит для них устойчивые признаки. Редкие классы, наоборот, могут давать высокие значения специфичности, но низкую полноту, поскольку положительных примеров недостаточно для надежного обобщения.

По этой причине при анализе ChestX-ray14 нельзя ограничиваться одной средней метрикой. В отчете по обучению нужно смотреть значения по каждому классу отдельно. Это помогает заранее увидеть, какие наблюдения

распознаются увереннее, а какие требуют дополнительной настройки данных, порогов или стратегии обучения.

Распределение классов в наборе ChestX-ray14 показано на рисунке 2.

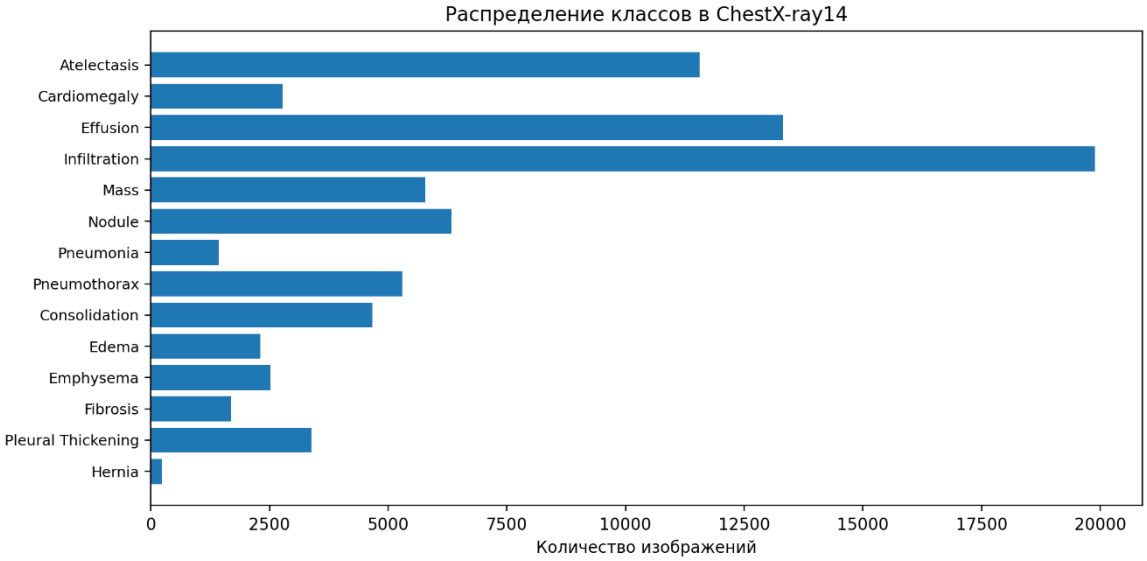


Рисунок 2 – Распределение классов в наборе ChestX-ray14

1.5 Разделение данных и качество разметки

При подготовке обучающей выборки важно избежать попадания снимков одного пациента одновременно в разные подвыборки. Если похожие снимки одного пациента окажутся и в обучении, и в тестировании, итоговая оценка качества может быть завышена. Поэтому при наличии идентификатора пациента разделение следует выполнять по patient_id.

Разметка ChestX-ray14 получена на основе медицинских описаний, поэтому ее нельзя считать полностью свободной от ошибок. В части записей возможны неточности, связанные с формулировками в исходных отчетах. Это не делает набор непригодным, но требует осторожности при интерпретации результатов обучения.

Для разрабатываемой системы имеет смысл использовать три части данных: обучающую, валидационную и тестовую. Обучающая часть нужна для настройки параметров модели, валидационная - для выбора лучшей версии модели, тестовая - для итоговой проверки качества.

Разделение данных фиксируется в подготовленном CSV-файле. Это сделано для того, чтобы обучение, оценка и анализ ошибок работали с одним

и тем же разбиением. Если каждый скрипт будет случайно делить данные заново, результаты разных запусков станет сложно сравнивать между собой.

При проверке разметки также важно учитывать строку No Finding. В программе она не используется как отдельный патологический класс, а переводится в отсутствие положительных меток. Такой подход сохраняет многометочную постановку и не смешивает нормальные снимки с набором целевых наблюдений.

Назначение подвыборок данных приведено в таблице 3.

Таблица 3 – Назначение подвыборок данных

Подвыборка	Назначение	Особенность использования
Train	Обучение параметров модели	Применяются аугментации и случайные преобразования
Validation	Контроль качества в процессе обучения	Используется для выбора лучшей эпохи
Test	Итоговая проверка	Не используется при обучении и подборе параметров

1.6 Предварительная обработка изображений

Перед передачей снимка в нейронную сеть изображение должно быть приведено к единому виду. Входные данные могут различаться по размеру, яркости, контрастности и формату. Без предварительной обработки модель будет получать неоднородные данные, что ухудшит устойчивость результата.

Для изображений PNG и JPG достаточно чтения файла как обычного изображения. Из DICOM берется только пиксельная матрица. Служебные поля, которые могут содержать данные пациента, не должны сохраняться в отчете или передаваться в интерфейс.

В разрабатываемом решении применяется следующий порядок подготовки: чтение файла, перевод в оттенки серого, приведение к трем каналам, изменение размера до 224 x 224 пикселей, нормализация и формирование тензора PyTorch. Трехканальный формат нужен потому, что ResNet50 с предобученными весами ожидает вход такого вида.

Этапы предварительной обработки снимка приведены в таблице 4.

Таблица 4 – Этапы предварительной обработки снимка

Этап	Операция	Результат
1	Проверка типа файла	Отклонение неподдерживаемых форматов
2	Чтение пиксельных данных	Получение изображения без персональных метаданных
3	Изменение размера	Приведение к входу 224 x 224
4	Нормализация	Стабилизация входных значений
5	Формирование тензора	Подготовка данных для инференса

Для обучающей выборки дополнительно используются аугментации: небольшой поворот, изменение масштаба и горизонтальное отражение. Для тестовой части такие случайные преобразования не применяются, поскольку итоговая проверка должна быть воспроизводимой.

Аугментации применяются только в разумных пределах. Для рентгенограмм нельзя использовать слишком сильные повороты или искажения, поскольку они могут сделать изображение не похожим на реальный медицинский снимок. Поэтому используются умеренные преобразования: небольшие повороты, изменение масштаба и горизонтальное отражение.

Для валидации и тестирования случайные преобразования отключены. Это помогает сравнивать результаты разных запусков и понимать, что изменение метрик связано с моделью или весами, а не со случайным изменением изображения на этапе проверки.

1.7 Выбор архитектуры ResNet50

Основной моделью выбрана ResNet50. Эта архитектура достаточно глубока для выделения сложных признаков изображения и при этом хорошо поддерживается в PyTorch и torchvision. Важным преимуществом - наличие предобученных весов, что дает возможность использовать перенос обучения [4, 5].

ResNet50 построена на остаточных блоках. Они помогают передавать информацию через несколько слоев и снижают риск ухудшения качества при

увеличении глубины сети. Для задачи рентгеновской классификации это удобно, поскольку модель должна учитывать не только отдельные локальные изменения, но и общую структуру снимка.

Последний классификационный слой заменяется на слой с четырнадцатью выходами. Каждый выход соответствует одному классу ChestX-ray14. Значения выходов обрабатываются независимо, поэтому модель может отметить несколько признаков на одном снимке.

Общая схема работы сверточной модели в выбранной постановке показана на рисунке 3.

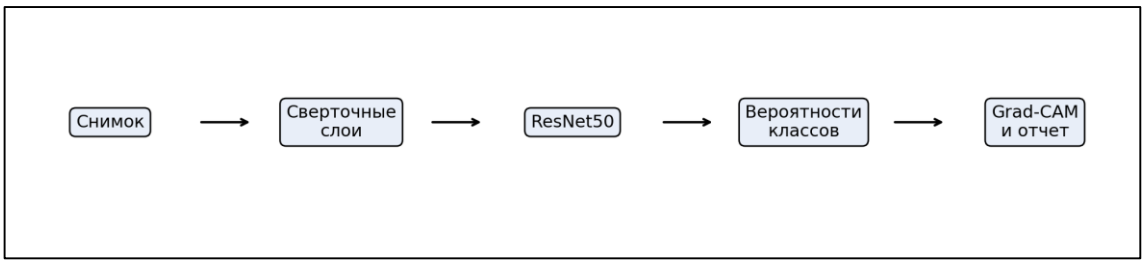


Рисунок 3 – Общая схема работы CNN-классификатора

1.8 Метрики оценки качества

Для медицинских изображений недостаточно оценивать модель только по доле правильных ответов. При дисбалансе классов модель может часто угадывать отрицательные случаи и при этом плохо находить редкие положительные примеры. Поэтому оценка должна включать несколько метрик.

В качестве основных метрик используются AUROC, precision, recall, F1-score, sensitivity и specificity. AUROC показывает способность модели разделять положительные и отрицательные примеры при разных порогах. Чувствительность показывает, насколько хорошо модель находит положительные случаи, а специфичность - насколько хорошо она не срабатывает на отрицательных случаях.

Используемые метрики оценки модели приведены в таблице 5.

Таблица 5 – Метрики оценки модели

Метрика	Что показывает	Почему важна
AUROC	Разделимость положительных и отрицательных примеров	Устойчива к выбору одного фиксированного порога
Precision	Доля верных положительных срабатываний	Помогает оценить количество ложных тревог
Recall / Sensitivity	Доля найденных положительных случаев	Важна при поиске признаков заболевания
Specificity	Доля правильно отклоненных отрицательных случаев	Показывает устойчивость к ложным срабатываниям
F1-score	Баланс precision и recall	Удобна при сравнении качества по классам

Метрики нужно считать отдельно по каждому классу, а затем формировать усредненную оценку. Такой подход помогает увидеть, по каким наблюдениям модель работает стабильно, а где нужна дополнительная настройка данных или порога решения.

Отдельная оценка по классам помогает обнаруживать ситуацию, когда среднее значение выглядит приемлемым, но один или несколько классов фактически не распознаются. Для медицинских изображений это особенно важно: редкий класс может занимать небольшую долю датасета, но быть значимым для анализа снимка.

В интерфейсе метрики выводятся вместе с порогами классов. Это помогает увидеть, что F1 и чувствительность зависят не только от качества признаков, но и от выбранного порога. Если AUROC достаточно высокий, а F1 низкий, это может говорить о необходимости настройки threshold, а не о полной непригодности модели.

1.9 Визуальное пояснение результата

Для повышения прозрачности результата в систему включается механизм Grad-CAM. Он показывает области изображения, которые сильнее

всего повлияли на выбранный выход модели. В интерфейсе тепловая карта отображается вместе со списком вероятностей.

Такая визуализация помогает обнаружить случаи, когда модель ориентируется не на область легких, а на подписи, рамки, края снимка или другие нерелевантные элементы. При работе с открытыми наборами данных это особенно заметно: изображения могут иметь разное происхождение и разное качество подготовки.

Grad-CAM нельзя воспринимать как точную разметку патологической области. Это инструмент для анализа поведения модели. Он полезен при проверке результата, но окончательная интерпретация медицинского снимка остается задачей специалиста.

1.10 Ограничения применения автоматической классификации

Автоматическая классификация медицинских снимков может использоваться как инструмент предварительного анализа, но не должна заменять врача. Даже хорошо обученная модель зависит от качества данных, разметки, оборудования и условий, в которых были получены изображения.

Для применения в реальном медицинском процессе нужна отдельная клиническая проверка. Нужно оценить модель на независимых данных, исследовать ошибки, проверить устойчивость к разным источникам изображений и убедиться, что тепловые карты не указывают на нерелевантные области.

В разрабатываемом программном комплексе результат сопровождается предупреждением о том, что он носит исследовательский характер. Это ограничение неформальность: оно задает правильный способ использования системы и снижает риск неверной трактовки автоматического вывода.

1.11 Вывод по разделу

В ходе анализа предметной области определены основные особенности задачи автоматической классификации рентгеновских снимков грудной клетки. Выбранный набор данных ChestX-ray14 подходит для обучения

многометочной модели, но требует аккуратной подготовки, раздельной оценки классов и учета дисбаланса.

Для построения программного комплекса обосновано применение ResNet50 с переносом обучения. Дополнительно определены требования к предварительной обработке изображений, расчету метрик, сохранению версии модели и визуальному пояснению результата. Эти решения задают основу для проектирования архитектуры приложения и последующей реализации.

2 Архитектура сверточных нейронных сетей

Сверточные нейронные сети применяются в задачах компьютерного зрения благодаря способности учитывать пространственную структуру изображения. В отличие от полносвязной сети, CNN не рассматривает пиксели как независимый набор чисел. Она анализирует локальные участки изображения и постепенно формирует признаки более высокого уровня.

При обработке рентгеновских снимков это свойство особенно важно. Изменения на снимке могут быть небольшими и располагаться в разных областях. Поэтому модель должна находить локальные признаки и одновременно учитывать общую картину изображения.

2.1 Представление изображения

Цифровое изображение можно рассматривать как матрицу значений яркости. Для цветных изображений используется несколько каналов, для рентгенограмм обычно достаточно одного яркостного канала. В выбранной реализации снимок приводится к трем каналам, чтобы его можно было передать в ResNet50 с предобученными весами.

После изменения размера и нормализации изображение превращается в тензор. Для одного снимка в режиме инференса используется форма batch x channels x height x width. Такой формат стандартен для PyTorch и удобен для пакетной обработки изображений.

Тензорное представление удобно тем, что оно не зависит от исходного формата файла. После предварительной обработки PNG, JPG и DICOM приводятся к единому виду. Это упрощает работу модели и дает возможность использовать один и тот же сервис инференса для пользовательских снимков и тестовой выборки.

Нормализация значений пикселей используется для согласования входа с предобученной моделью. ResNet50 обучалась на изображениях с определенным распределением каналов, поэтому входные данные приводятся к ожидаемому диапазону. Без этого выходы модели могут стать менее устойчивыми, особенно при работе с изображениями из разных источников.

2.2 Сверточный слой

Сверточный слой применяет к изображению набор небольших фильтров. Каждый фильтр перемещается по изображению и формирует карту признаков. На ранних слоях обычно выделяются границы, контуры и перепады яркости. На более глубоких слоях признаки становятся более сложными.

Для рентгеновских снимков ранние признаки также имеют значение. Контуры ребер, границы легочных полей и переходы яркости формируют основу для более сложных представлений. На глубоких слоях модель уже не работает с отдельными пикселями, а использует комбинации признаков, которые могут соответствовать областям затемнения, уплотнения или изменения формы.

Сверточный слой удобен тем, что один и тот же фильтр применяется ко всем участкам изображения (3). Благодаря этому модель может находить похожие признаки независимо от их положения. Для рентгенограмм это важно, поскольку изменения могут находиться в разных частях легочных полей.

$$Y(i,j) = \sum_m \sum_n X(i+m,\ j+n)K(m,n) \qquad (3)$$

где X – входное изображение;

K – ядро свертки;

Y – полученная карта признаков.

В работе достаточно использовать эту формулу как общее описание операции свертки; остальные параметры, такие как шаг и отступ, удобнее рассматривать на уровне настройки архитектуры.

2.3 Функции активации, подвыборка и нормализация

После свертки используется функция активации. В современных CNN чаще всего применяется ReLU, которая обнуляет отрицательные значения и

оставляет положительные. Это делает сеть нелинейной и позволяет ей описывать более сложные зависимости в данных.

Слои подвыборки уменьшают размер карты признаков. За счет этого снижается вычислительная нагрузка, а модель становится менее чувствительной к небольшим смещениям изображения. В ResNet также используются нормализация и остаточные связи, которые повышают устойчивость обучения глубоких сетей.

2.4 Остаточные блоки ResNet50

ResNet50 отличается использованием остаточных блоков. В таком блоке результат преобразования складывается с исходным входом блока (4). Это помогает передавать информацию через сеть напрямую и уменьшает проблему деградации качества при увеличении глубины модели:

$$Y = F(X) + X \tag{4}$$

где X – вход блока;

$F(X)$ – преобразование внутри блока;

Y – выход блока.

Простая форма записи отражает основную идею ResNet: сеть обучается не полностью новому отображению, а добавочному изменению к входным признакам.

Для задачи классификации снимков грудной клетки ResNet50 используется как извлекатель признаков. Последний слой заменяется на новый классификатор, рассчитанный на четырнадцать выходов.

Выбор ResNet50 выступает компромиссом между глубиной модели и сложностью ее использования. Более простые сети могут не выделять достаточно сложные признаки, а чрезмерно тяжелые архитектуры требуют больше вычислительных ресурсов. ResNet50 хорошо поддерживается в

PyTorch, за счет этого проще загрузку предобученных весов и замену последнего слоя.

Остаточные связи особенно полезны при дообучении. При переносе обучения часть признаков уже сформирована, и модель должна адаптировать их к рентгенограммам. Остаточные блоки помогают сохранять полезную информацию на разных глубинах сети и уменьшают риск нестабильного обучения.

2.5 Многометочный выход модели

В обычной многоклассовой классификации применяется softmax, поскольку модель выбирает один класс. Для ChestX-ray14 такой подход не подходит: несколько наблюдений могут присутствовать одновременно. Поэтому каждый выход модели обрабатывается независимо с помощью сигмоидальной функции (5):

$$p_i = \sigma(z_i) = \frac{1}{1+e^{-z_i}} \quad (5)$$

где z_i - выход модели для i-го класса;

p_i – вероятность наличия соответствующего признака.

После получения вероятностей применяется порог решения. В базовом варианте используется 0,5, но после обучения пороги могут настраиваться отдельно по классам.

2.6 Перенос обучения

Обучение глубокой CNN с нуля требует большого количества данных и вычислительных ресурсов. В рассматриваемой задаче используется перенос обучения: модель берет начальные признаки из ResNet50, обученной на ImageNet, а затем донастраивается на рентгеновских снимках.

Такой подход не означает, что признаки обычных изображений полностью совпадают с медицинскими. Однако ранние слои, выделяющие

контуры и простые текстуры, могут быть полезны и для рентгенограмм. Это сокращает время обучения и делает процесс устойчивее.

В практической реализации можно использовать разные режимы переноса обучения. На первом этапе возможно обучать только последний слой, чтобы быстро проверить работоспособность контура. Затем можно разморозить часть глубоких слоев и дообучить модель более точно. В проекте такая возможность сохраняется за счет отдельного модуля построения модели.

Перенос обучения также снижает требования к времени эксперимента. Полное обучение глубокой CNN с нуля требует значительно больше данных и вычислений. Для исследовательского прототипа разумнее использовать предобученную основу и сосредоточиться на корректной подготовке данных, оценке качества и проверке поведения модели.

2.7 Оценка качества CNN

После обучения модель проверяется на тестовой выборке. Для каждого класса рассчитываются отдельные метрики, потому что качество по редким классам может заметно отличаться от качества по часто встречающимся признакам. Итоговая оценка должна показывать не только средний результат, но и слабые места модели.

При анализе результатов особое внимание уделяется AUROC, чувствительности и специфичности. Эти показатели позволяют оценить работу модели с учетом разных порогов и соотношения положительных и отрицательных случаев.

2.8 Grad-CAM в структуре CNN

Grad-CAM используется после получения результата классификации. Метод берет выбранный класс и анализирует, какие области последнего сверточного слоя оказали влияние на этот выход. Затем строится тепловая карта, которая масштабируется до размера исходного изображения.

Для пользователя тепловая карта не отдельный диагноз, а пояснение поведения модели. По ней видно, на какие зоны снимка модель опиралась при формировании результата.

2.9 Вывод по разделу

Архитектура CNN подходит для анализа рентгеновских снимков, поскольку позволяет извлекать признаки изображения последовательно и учитывать их пространственное расположение. ResNet50 выбрана как базовая модель благодаря остаточным связям, наличию предобученных весов и поддержке в PyTorch.

Для выбранной задачи модель настраивается на многометочный выход. Это дает возможность получать вероятность каждого наблюдения отдельно. Дополнение в виде Grad-CAM делает результат более понятным и помогает выявлять ошибки поведения модели.

3 Анализ аналогов

Перед проектированием собственного программного комплекса нужно рассмотреть существующие решения, которые связаны с автоматическим анализом рентгеновских снимков. В этой области можно выделить исследовательские модели, специализированные библиотеки и коммерческие системы поддержки анализа медицинских изображений.

Сравнение аналогов важно не для прямого копирования их функций, а для понимания того, какие возможности уже применяются на практике и какие ограничения остаются актуальными. Для разрабатываемой системы особенно важны поддержка рентгенограмм, наличие объяснимости, возможность обучения на собственных данных и открытость программной части.

3.1 Критерии сравнения

Для анализа выбраны критерии, которые связаны с задачей классификации медицинских снимков: открытость решения, поддержка рентгеновских изображений, возможность использовать модель в исследовательском контуре, наличие визуального пояснения и удобство интеграции в приложение.

Критерии сравнения аналогов приведены в таблице 6.

Таблица 6 – Критерии сравнения аналогов

Критерий	Содержание
Открытость	Возможность изучать модель, код или описание метода
Поддержка рентгенограмм	Оёриентация на снимки органов грудной клетки
Объяснимость	Наличие тепловых карт или других пояснений
Интеграция	Возможность использовать в собственной системе
Обучение	Возможность дообучения или проверки на своем наборе данных

3.2 CheXNet

CheXNet является одним из известных исследовательских примеров применения глубокого обучения к рентгенограммам грудной клетки. В этой

работе использовалась глубокая сверточная сеть для выявления пневмонии на снимках [6]. Данный пример показывает, что CNN могут применяться к медицинским изображениям при наличии крупного набора данных и корректной проверки качества.

Сильной стороной CheXNet является ориентация на конкретную медицинскую задачу и использование реальных рентгенограмм. Ограничение состоит в том, что это исследовательская модель, а не готовое приложение с интерфейсом, хранением отчетов и обработкой пользовательских файлов.

Для рассматриваемого проекта CheXNet полезен как ориентир по постановке задачи. Он показывает, что для рентгенограмм важны крупные датасеты, аккуратная проверка метрик и сопоставление результата с экспертной оценкой. При этом прямое использование такой модели не решает задачу создания собственного приложения с загрузкой снимков и хранением отчетов.

Отличие разрабатываемой системы состоит в том, что в ней внимание уделено не только самой модели, но и полному программному маршруту. Пользователь получает интерфейс, API, сохраненный отчет и визуальное пояснение, а не только результат работы нейросетевой модели.

3.3 TorchXRayVision

TorchXRayVision – это библиотеку и набор моделей для работы с рентгеновскими изображениями [7]. Она полезна для проведения экспериментов, поскольку содержит готовые инструменты и модели, ориентированные именно на медицинские снимки.

Для прикладного решения библиотеку нужно дополнять собственным API, интерфейсом, подсистемой отчетов и контролем входных данных. Поэтому TorchXRayVision можно рассматривать как полезный источник инструментов, но не как готовую систему для конечного пользователя.

TorchXRayVision может быть полезен при сравнении с готовыми медицинскими моделями, но в данном проекте выбран самостоятельный контур на PyTorch и ResNet50. Это помогает явно показать все этапы:

подготовку ChestX-ray14, обучение, сохранение весов, оценку качества и подключение модели к backend.

Так решение становится прозрачнее. Если использовать готовую библиотеку как черный ящик, часть этапов подготовки и обучения остается скрытой. В собственной реализации проще объяснить, какие данные поступают на вход модели и какие файлы формируются после эксперимента.

3.4 MONAI

MONAI – это фреймворк для медицинского глубокого обучения, который включает средства подготовки данных, аугментаций, обучения и инференса [8]. Он применяется в разных задачах медицинской визуализации, включая КТ, МРТ и рентгеновские снимки.

Преимущество MONAI заключается в широком наборе инструментов. При этом для рассматриваемого прототипа использование всего фреймворка может быть избыточным, поскольку задача сосредоточена на классификации рентгенограмм с одной выбранной архитектурой ResNet50.

MONAI хорошо подходит для крупных медицинских проектов, где нужна работа с разными типами данных и сложными сценариями обработки. Однако для выбранной задачи избыточность фреймворка может усложнить структуру проекта. В работе используется более компактный набор библиотек, достаточный для классификации рентгенограмм и построения Grad-CAM.

При этом отдельные идеи из подобных фреймворков учитываются при проектировании: разделение preprocessing, модели, метрик и инференса на самостоятельные модули. Это помогает сохранить понятную архитектуру без подключения большого числа зависимостей.

3.5 Коммерческие системы анализа снимков

На рынке существуют коммерческие системы поддержки анализа рентгенологических исследований. К ним относятся решения, которые помогают врачам находить признаки патологий, сортировать исследования и ускорять работу с большим потоком снимков. Обычно такие системы имеют

готовый интерфейс, механизмы интеграции с медицинскими информационными системами и обученные модели.

Главное ограничение коммерческих решений для разрабатываемой системы связано с закрытостью. Пользователь не может свободно изменить модель, проверить обучение на собственном наборе данных или изучить детали подготовки признаков. Для исследовательского программного комплекса важнее контролируемая архитектура и возможность воспроизводимой проверки результатов.

Коммерческие CAD-системы обычно ориентированы на внедрение в медицинскую инфраструктуру и требуют сертификации, настройки интеграции и контроля со стороны производителя. В разработке исследовательского комплекса такой путь не подходит, поскольку важно иметь возможность менять код, обучать модель заново и проверять результаты на собственном наборе данных.

В то же время коммерческие решения показывают, какие функции ожидаются от прикладной системы: понятный интерфейс, история исследований, визуальная подсказка и стабильная обработка ошибок. Эти особенности учтены в архитектуре MedVision CNN, но без заявления о клиническом назначении системы.

3.6 Сравнительный анализ

По итогам сравнения видно, что существующие решения закрывают разные части задачи. Исследовательские проекты помогают выбрать метод и понять ожидаемые трудности, библиотеки ускоряют эксперименты, а коммерческие системы демонстрируют прикладной сценарий использования. Разрабатываемое приложение объединяет эти подходы в ограниченном, но контролируемом виде: оно должно включать загрузку снимка, классификацию, визуальное пояснение и сохранение результата.

По итогам анализа аналогов выбран вариант собственной реализации. Он позволяет не зависеть от закрытого сервиса и одновременно сохранить

прикладной сценарий работы через браузер. Для проверки ResNet50 и оценки поведения модели на ChestX-ray14 такой путь подходит лучше.

Кроме того, собственная реализация дает возможность расширять систему постепенно. В дальнейшем можно добавить другую архитектуру, изменить стратегию обработки DICOM, подключить PostgreSQL или реализовать экспорт отчета, не меняя весь программный комплекс.

Сравнение аналогов приведено в таблице 7.

Таблица 7 – Сравнение аналогов

Решение	Тип	Сильные стороны	Ограничения
CheXNet	Исследовательская модель	Показывает применимость CNN к рентгенограммам	Не является готовым web-приложением
TorchXRayVision	Библиотека	Есть модели и инструменты для рентгеновских снимков	Требуется самостоятельная интеграция
MONAI	Фреймворк	Широкий набор средств для медицинских данных	Может быть избыточен для узкого прототипа
Коммерческие CAD-системы	Прикладные продукты	Готовые интерфейсы и интеграция	Закрытая логика и ограниченная изменяемость

4 Архитектура приложения

Архитектура приложения должна обеспечивать полный путь обработки снимка: загрузку файла, предварительную подготовку изображения, запуск классификатора, построение визуального пояснения и сохранение результата. При проектировании важно разделить эти действия на самостоятельные компоненты, чтобы в дальнейшем можно было менять модель или способ хранения без полной переработки системы.

4.1 Общая схема

Приложение проектируется как web-система. Пользователь работает через браузер, а все вычисления выполняются на стороне сервера. Такой вариант удобен для демонстрации, тестирования и дальнейшего расширения, поскольку клиентская часть не содержит тяжелой логики машинного обучения.

Архитектура программного комплекса показана на рисунке 4.

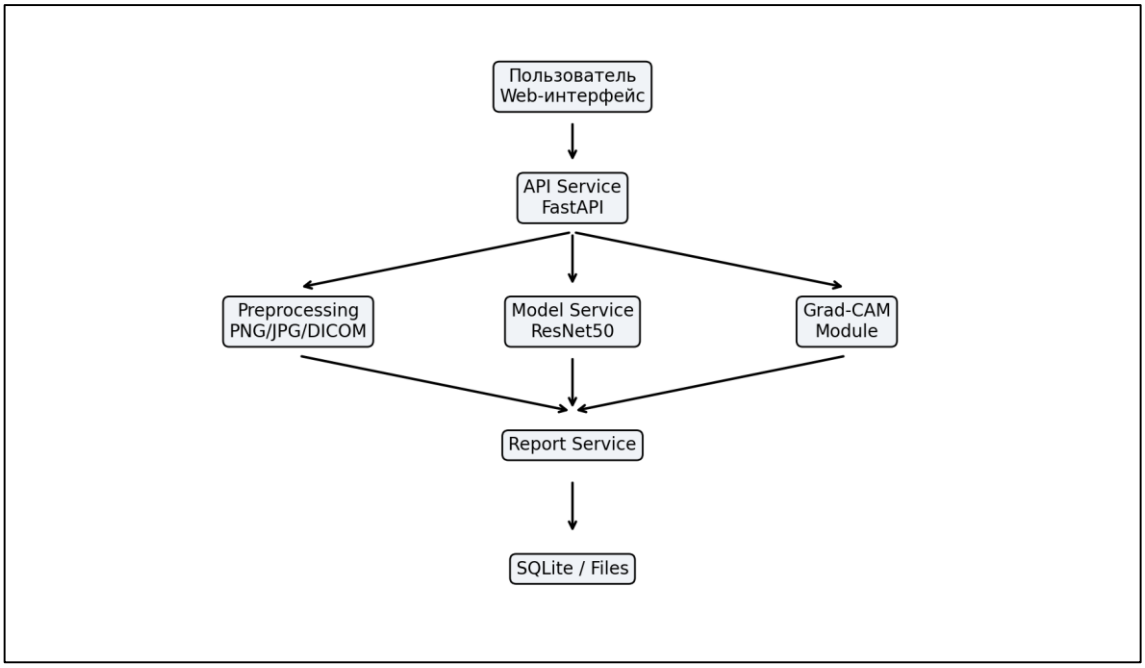


Рисунок 4 – Архитектура программного комплекса

Основными компонентами являются клиентская часть, API Service, модуль предварительной обработки, сервис классификации ResNet50, модуль Grad-CAM, подсистема отчетов и хранилище файлов. Взаимодействие между клиентом и сервером выполняется через REST API [9].

Взаимодействие компонентов строится последовательно. Сначала клиент отправляет файл на сервер, затем API вызывает preprocessing, после этого выполняется инференс модели и строится Grad-CAM. В конце подсистема отчетов сохраняет результат, а клиент получает ссылки на изображения и структурированные данные.

Такой маршрут удобен для тестирования. Если возникает ошибка, можно определить, на каком этапе она произошла: при чтении файла, подготовке изображения, запуске модели, построении тепловой карты или сохранении отчета. Это упрощает отладку и делает систему более надежной.

4.2 Клиентская часть

Клиентская часть отвечает за загрузку снимка и отображение результата. В интерфейсе размещается область выбора файла, предупреждение о назначении системы, кнопка запуска анализа, блок вероятностей по классам и изображение Grad-CAM.

Для реализации интерфейса имеет смысл использовать React и TypeScript. Компонентный подход позволяет разделить страницу на отдельные части: UploadPanel, ResultCard, GradCamView и HistorySidebar. Такая структура упрощает сопровождение интерфейса и дальнейшее добавление новых экранов.

В интерфейсе важно не перегружать пользователя техническими деталями работы модели. Поэтому основные данные показываются в виде нескольких блоков: карточки загруженного изображения, списка вероятностей, блока выбранных классов и области с визуальным пояснением. Такой подход помогает быстро увидеть результат анализа и при необходимости перейти к более подробной информации о метриках обучения и версии модели.

Для поддержки воспроизводимости пользователь видит не только названия найденных классов, но и численные вероятности, пороги принятия решения и идентификатор отчета. Это связывает экран приложения с

сохраненной записью в базе данных и упрощает повторную проверку результата.

4.3 Серверная часть и API

Серверная часть реализуется на FastAPI. Этот фреймворк удобен для задач, где нужно принимать файлы, возвращать JSON-ответы и вызывать Python-модули машинного обучения [10]. В приложении API Service служит входной точкой для всех пользовательских запросов.

Основные маршруты API приведены в таблице 8.

Таблица 8 – Основные маршруты API

Маршрут	Метод	Назначение
/health	GET	Проверка доступности сервиса
/api/v1/classify	POST	Загрузка снимка и запуск классификации
/api/v1/reports	GET	Получение списка отчетов
/api/v1/reports/{id}	GET	Получение результата анализа
/api/v1/reports/{id}/gradcam	GET	Получение изображения Grad-CAM

Ответ классификации должен содержать идентификатор отчета, версию модели, список выбранных меток, вероятности по всем классам, ссылку на Grad-CAM и предупреждение о назначении результата.

Маршруты API не выполняют всю обработку самостоятельно, а выступают как слой координации. После получения файла сервер вызывает сервис предварительной обработки, затем передает тензор в сервис модели, запускает построение Grad-CAM и только после этого создает запись отчета. Такой порядок делает код более понятным: каждый модуль отвечает за собственный этап обработки.

Отдельное внимание уделено ошибкам. Неподдерживаемый формат, слишком большой файл, отсутствие пиксельных данных в DICOM и недоступность модели должны возвращаться пользователю в виде понятного JSON-ответа. Это особенно важно при тестировании приложения, поскольку ошибки входных данных встречаются чаще, чем ошибки самой модели.

4.4 Модуль предварительной обработки

Модуль предварительной обработки изолирует работу с изображениями от остальной части приложения. Он принимает содержимое файла, определяет формат, читает пиксельные данные и формирует тензор для модели. Для PNG и JPG используется Pillow, для DICOM - pydicom [11].

При обработке DICOM-файлов в отчет не передаются персональные поля. В результате сохраняются только служебные сведения, необходимые для воспроизводимости анализа: имя файла в безопасном виде, версия модели, вероятности классов и путь к тепловой карте.

Изоляция preprocessing в отдельном сервисе снижает риск ошибок в других частях приложения. Backend-маршрут не должен знать, как именно читается DICOM или применяется нормализация. Он получает готовый тензор и изображение для отображения, а вся логика подготовки сосредоточена в одном месте.

При изменении способа обработки достаточно изменить один модуль. Например, если в дальнейшем будет добавлено выравнивание гистограммы или обрезка области легких, сервис модели и клиентская часть останутся прежними. Это соответствует выбранной модульной архитектуре.

Такой подход удобен при последующем сравнении вариантов preprocessing. Например, можно отдельно проверить влияние выравнивания яркости, изменения размера или нормализации, не меняя код API и пользовательского интерфейса. В результате экспериментальная часть отделяется от прикладной части приложения.

Важным моментом остается сохранение изображения для отображения пользователю. Модель получает нормализованный тензор, но в отчете нужно показывать снимок в привычном визуальном виде. Поэтому сервис возвращает не только тензор, но и изображение, пригодное для сохранения и последующего наложения Grad-CAM.

4.5 Сервис классификации

Сервис классификации загружает ResNet50 при запуске приложения и использует ее в режиме eval. Это помогает не загружать модель заново при каждом запросе. Если обученные веса отсутствуют, система может работать в демонстрационном режиме, но интерфейс и документация должны явно указывать на это ограничение.

После инференса выходы модели преобразуются в вероятности. Далее применяется порог решения, и формируется список меток, которые будут отображены пользователю. Для сохранения воспроизводимости в каждом отчете фиксируется имя модели и версия весов.

Версия модели хранится вместе с отчетом не случайно. После повторного обучения веса могут измениться, а значит один и тот же снимок может получить немного другой набор вероятностей. Если в отчете указана версия эксперимента, результат можно связать с конкретным файлом весов, конфигурацией обучения и порогами классов.

В сервисе классификации модель переводится в режим eval. Это отключает поведение, характерное только для обучения, и делает инференс стабильным. Запросы пользователей обрабатываются уже загруженной моделью, поэтому время ответа не увеличивается за счет повторной инициализации ResNet50.

4.6 Модуль Grad-CAM

Модуль Grad-CAM вызывается после классификации. Он выбирает класс с наибольшей вероятностью или класс, выбранный пользователем, строит тепловую карту и сохраняет изображение overlay. Это изображение отображается в интерфейсе рядом с таблицей вероятностей.

Такое разделение удобно: классификатор отвечает только за вероятности, а модуль объяснимости - за визуальную интерпретацию результата. При необходимости метод объяснимости можно заменить, не меняя основной маршрут классификации.

Визуальное пояснение сохраняется как отдельное изображение, а не строится заново при каждом открытии отчета. Это уменьшает нагрузку на сервер и делает просмотр истории быстрее. В базе данных хранится только путь к файлу, а само изображение размещается в файловом хранилище вместе с исходным снимком.

Такой вариант удобен и при последующем анализе ошибок. Если тепловая карта указывает на маркер, подпись или край изображения, можно вернуться к исходному снимку и проверить, не связан ли результат с особенностями набора данных или этапа предварительной обработки.

4.7 Хранение отчетов

Для локального прототипа достаточно SQLite. В базе данных сохраняются метаданные анализа: идентификатор отчета, дата создания, исходное имя файла в безопасном виде, версия модели, вероятности, выбранные метки и путь к Grad-CAM. Сами изображения удобнее хранить в файловой системе.

Структура сохраняемого отчета приведена в таблице 9.

Таблица 9 – Структура сохраняемого отчета

Поле	Назначение
id	Уникальный идентификатор отчета
created_at	Дата и время анализа
model_version	Версия использованных весов
probabilities_json	Вероятности по классам
labels_json	Выбранные метки
gradcam_path	Путь к изображению Grad-CAM
warning	Текст предупреждения о назначении результата

Такой подход упрощает дальнейшее развитие системы. При росте объема данных SQLite можно заменить на PostgreSQL, а локальное файловое хранилище - на объектное хранилище, не меняя логику классификации.

Структура отчета также помогает при анализе результатов обучения. Если несколько версий модели дают разные ответы на один и тот же снимок, сохраненные поля позволяют сравнить модель, пороги и вероятности. В

дальнейшем на этой основе можно добавить сравнение отчетов или повторную обработку снимка новой версией весов.

Для файлового хранилища выбран простой вариант с локальными каталогами. Он удобен на этапе разработки и демонстрации. При увеличении количества данных его можно заменить на отдельное объектное хранилище, не меняя логику сохранения метаданных в базе.

4.8 Обработка ошибок

Система должна корректно реагировать на неподдерживаемый формат, поврежденный файл, отсутствие модели и ошибки чтения DICOM. Пользователь должен получать понятное сообщение, а не технический стек ошибки. Это важно для удобства работы и для последующего тестирования приложения.

Основные ошибки и реакция системы приведены в таблице 10.

Таблица 10 – Основные ошибки и реакция системы

Ситуация	Реакция
Неподдерживаемый формат	HTTP 415 и сообщение о допустимых форматах
Файл не читается	HTTP 400 и сообщение о поврежденном изображении
Модель не загружена	HTTP 500 с пояснением причины
Не удалось построить Grad-CAM	Сохранение отчета без overlay и сообщение пользователю

5 Реализация приложения

После выбора архитектуры приложения следующим этапом стала программная реализация комплекса MedVision CNN. В данном разделе основное внимание уделено конкретным модулям проекта: подготовке ChestX-ray14, обработке изображений, обучению ResNet50, API, Grad-CAM и клиентской части. Программный код приведен только в тех местах, где он помогает показать устройство решения.

В реализации сохранен главный принцип, заложенный на этапе проектирования: модель и пользовательский интерфейс не смешиваются в одном модуле. За счет этого можно отдельно изменять алгоритм предварительной обработки, переобучать ResNet50, обновлять пороги классов или дорабатывать интерфейс без полной переработки приложения.

Проект построен вокруг датасета ChestX-ray14. Он используется для подготовки обучающей выборки, обучения модели и последующей проверки качества. Для пользовательского режима также поддерживаются отдельные изображения в форматах PNG, JPG и DICOM. При этом DICOM-файлы используются только как источник пиксельных данных, без сохранения персональных полей пациента.

5.1 Подготовка датасета ChestX-ray14

Для обучения используется исходный CSV-файл Data_Entry_2017.csv из ChestX-ray14. В нем каждая строка соответствует одному снимку, а разметка хранится в поле Finding Labels. Так как у одного изображения может быть несколько признаков, метки разделяются символом «|». В приложении эта строка преобразуется в бинарный вектор из 14 элементов, соответствующий классам датасета.

Подготовка выполняется скриптом app.ml.prepare_chestxray14. Скрипт читает исходную таблицу, разбирает поле с диагнозами, формирует отдельные бинарные колонки, проверяет наличие PNG-файлов и сохраняет новый CSV. Строки с меткой No Finding переводятся в нулевой вектор по всем патологическим классам. Такой вариант соответствует выбранной

многометочной постановке, где отсутствие перечисленных находок не считается отдельным патологическим классом.

В скрипте подготовки датасета разбор поля Finding Labels выполняется следующим образом:

```
def parse_labels(raw_labels: str) -> set[str]:
    labels = {label.strip() for label in
str(raw_labels).split("|") if label.strip()}
    if labels == {NO_FINDING_LABEL}:
        return set()
    return {label for label in labels if label in CLASSES}
label_values = {
    class_name: int(class_name in labels)
    for class_name in CLASSES
}
```

При подготовке учитывается patient_id. Если этот идентификатор доступен, разделение на train, validation и test выполняется по пациентам. Это важно: снимки одного пациента не должны одновременно попадать в обучающую и тестовую выборку, иначе оценка качества может быть завышена. Если patient_id не найден, программа использует разделение по изображениям и выводит предупреждение.

После перемешивания ключей формируются три непересекающиеся группы. В каждую строку подготовленного CSV добавляется поле split, по которому затем создаются DataLoader для обучения, валидации и тестирования. Такой вариант удобен тем, что одно и то же разбиение используется на всех этапах эксперимента и не пересоздается случайно при каждом запуске.

В подготовленном CSV каждая строка содержит путь к изображению, идентификатор пациента, значения четырнадцати классов и название подвыборки. Благодаря этому скрипты обучения и оценки не зависят от исходного формата Data_Entry_2017.csv и работают с уже нормализованной таблицей.

При таком формате таблицы удобно выполнять повторяемые эксперименты. Веса модели, пороги и метрики можно связать с конкретной версией prepared CSV, а значит при необходимости можно вернуться к тем же

данным и проверить результат заново. Это особенно важно при изменении параметров обучения, когда нужно понять, улучшилась ли модель из-за настройки или из-за случайного изменения разбиения.

В процессе подготовки также исключаются записи, для которых не найден файл изображения. Если такие строки оставить, ошибка возникнет уже во время обучения, и поиск причины займет больше времени. Поэтому проверка наличия файлов выполняется заранее, а итоговая статистика помогает увидеть, сколько записей было отброшено.

Такой промежуточный файл упрощает повторные эксперименты. Если потребуется изменить параметры обучения или архитектуру модели, не нужно заново разбирать исходную разметку. Достаточно использовать уже подготовленный CSV и зафиксированное разделение данных.

Дополнительно скрипт сохраняет статистику по классам. Она нужна не только для отчета, но и для проверки корректности подготовки данных: если после разборки разметки какой-либо класс неожиданно получил нулевое количество примеров, это сразу указывает на ошибку в названии метки или пути к изображениям.

```
if dataframe["patient_id"].notna().all():
    keys = dataframe["patient_id"].astype(str).unique()
    split_strategy = "patient_id"
else:
    keys = dataframe["image_path"].astype(str).unique()
    split_strategy = "image"

rng = np.random.default_rng(seed)
rng.shuffle(keys)
```

Результаты подготовки данных приведены в таблице 11.

Таблица 11 – Результаты подготовки данных

Этап	Что выполняется	Результат
Чтение исходного CSV	загрузка Data_Entry_2017.csv	получение списка снимков и строк Finding Labels
Разбор меток	выделение 14 патологических классов	бинарные колонки по каждому классу

Продолжение таблицы 11.

Этап	Что выполняется	Результат
Проверка файлов	поиск PNG в папке data/images	исключение строк без доступного изображения
Разделение данных	train/val/test с учетом patient_id	подготовленный набор для обучения и проверки

5.2 Реализация загрузки и предварительной обработки снимков

В серверной части предварительная обработка вынесена в отдельный сервис. Для обычных изображений используется Pillow, для DICOM-файлов - pydicom. При чтении DICOM сервис извлекает пиксельную матрицу, применяет rescale slope и rescale intercept при их наличии, приводит диапазон значений к 0-255 и учитывает вариант MONOCHROME1, где яркость изображения должна быть инвертирована.

Проверка расширения и чтение входного файла вынесены в отдельные функции:

```
def validate_extension(filename: str) -> str:
    extension = Path(filename).suffix.lower()
    if extension not in ALLOWED_EXTENSIONS:
        raise HTTPException(status_code=415, detail="Unsupported
file format")
    return extension
def load_image(filename: str, content: bytes):
    extension = validate_extension(filename)
    if extension in IMAGE_EXTENSIONS:
        image = Image.open(io.BytesIO(content))
        display_image =
ImageOps.exif_transpose(image).convert("L").convert("RGB")
    elif extension in DICOM_EXTENSIONS:
        display_image = _load_dicom_pixels(content)
    tensor = _transform(display_image).unsqueeze(0)
    return tensor, display_image
```

Для пользовательского инференса изображение приводится к формату RGB, хотя рентгенограмма является полутоновым изображением. Такое преобразование нужно для совместимости с ResNet50, предварительно обученной на ImageNet, где вход имеет три канала. После этого выполняется resize до 224 x 224 и нормализация по параметрам ImageNet. Аналогичный набор преобразований применяется при валидации и тестировании модели.

При обработке DICOM-файла используется только пиксельная матрица изображения:

```
dataset = pydicom.dcmread(io.BytesIO(content), force=True)
if "PixelData" not in dataset:
    raise HTTPException(status_code=400, detail="DICOM file has
no pixel data")

pixels = dataset.pixel_array.astype(np.float32)
pixels -= np.min(pixels)
if float(np.max(pixels)) > 0:
    pixels /= float(np.max(pixels))

pixels = (pixels * 255).astype(np.uint8)
return Image.fromarray(pixels).convert("L").convert("RGB")
```

Поддерживаемые форматы входных данных приведены в таблице 12.

Таблица 12 – Поддерживаемые форматы входных данных

Формат	Обработка	Примечание
PNG, JPG, JPEG	чтение через Pillow, учет EXIF-поворота, перевод в grayscale и RGB	используются для web-инференса и работы с ChestX-ray14
DICOM	чтение через pydicom, извлечение PixelData, нормализация яркости	персональные поля не сохраняются
Другие форматы	отклонение запроса	возвращается ошибка 415

Валидация файла выполняется до запуска модели. Если расширение не поддерживается, API возвращает ошибку 415. Если файл поврежден или не содержит пиксельных данных, возвращается ошибка 400. Такой подход позволяет не запускать инференс для некорректных запросов и не сохранять неполные отчеты.

5.3 Реализация модели ResNet50

Классификатор реализован на PyTorch. В качестве базовой архитектуры используется ResNet50 с предобученной основой. Последний слой заменяется на линейный слой с 14 выходами, так как ChestX-ray14 содержит 14 патологических классов. Задача не сводится к выбору одного класса, поэтому выходы модели рассматриваются независимо друг от друга.

Модель ResNet50 создается с заменой последнего классификационного слоя:

```
def build_resnet50(num_classes: int, pretrained: bool = True) ->
nn.Module:
    weights = models.ResNet50_Weights.DEFAULT if pretrained else
None
    try:
        model = models.resnet50(weights=weights)
        model.loaded_imagenet_weights = bool(weights is not
None)
    except Exception:
        model = models.resnet50(weights=None)
        model.loaded_imagenet_weights = False

    in_features = model.fc.in_features
    model.fc = nn.Linear(in_features, num_classes)
    return model
```

Во время инференса к логитам применяется sigmoid, после чего получается вероятность каждого класса. Для выбора итоговых меток используются индивидуальные пороги из файла thresholds.json. Такой вариант лучше общего порога 0,5, поскольку классы в ChestX-ray14 распределены неравномерно. Если thresholds.json отсутствует, сервис использует резервное значение из конфигурации.

Инференс модели и выбор итоговых меток выполняются в сервисе следующим образом:

```
with torch.no_grad():
    logits = self.model(input_tensor)
    probabilities_tensor =
torch.sigmoid(logits).detach().cpu()[0]

probabilities = {
    class_name: round(float(probabilities_tensor[index]), 4)
    for index, class_name in enumerate(CLASSES)
}

labels = [
    item["label"] for item in top_probabilities
    if item["selected"]
]
```

Вместо отдельной математической записи в реализации используется простое сравнение вероятности класса с индивидуальным порогом из файла thresholds.json. Это решение перенесено в программный код: backend получает

вектор вероятностей, сортирует классы по убыванию и помечает те из них, которые прошли заданный порог.

5.4 Обучение модели

Обучение запускается отдельным скриптом app.ml.train. На вход ему передаются путь к подготовленному CSV, каталог изображений, количество эпох, размер batch, learning rate и папка для сохранения эксперимента. В процессе обучения используется AdamW, а функция потерь BCEWithLogitsLoss получает параметр pos_weight. Он нужен для учета дисбаланса классов: ошибки по редким классам получают больший вес.

Один проход обучения по train loader реализован следующим образом:

```
def run_train_epoch(model, loader, criterion, optimizer,
device):
    model.train()
    total_loss = 0.0
    total_examples = 0
    for images, labels in loader:
        images = images.to(device)
        labels = labels.to(device)
        optimizer.zero_grad(set_to_none=True)
        logits = model(images)
        loss = criterion(logits, labels)
        loss.backward()
        optimizer.step()
        total_loss += float(loss.item()) * images.size(0)
        total_examples += images.size(0)
    return total_loss / max(total_examples, 1)
```

В конце каждой эпохи выполняется проверка на validation split. Скрипт рассчитывает loss, macro AUROC и macro F1. Лучшая модель сохраняется по значению macro AUROC. Вместе с весами сохраняются конфигурация эксперимента, список классов, журнал обучения, метрики на validation и график кривых обучения.

Кривые обучения используются как быстрый способ проверить поведение модели. Если train loss уменьшается, а validation loss растет, это указывает на переобучение. Если macro AUROC почти не меняется, то дальнейшее увеличение числа эпох может быть малоэффективным без изменения данных, аугментаций или learning rate.

В проведенном эксперименте значения train loss снижались на протяжении обучения, при этом validation loss постепенно увеличивался. Такая картина показывает, что модель хорошо подстраивается под обучающую выборку, но требует аккуратной настройки регуляризации и проверки качества на независимом наборе.

При такой динамике нельзя делать вывод только по одной кривой loss. Снижение train loss показывает, что модель подстраивается под обучающую выборку, но рост validation loss указывает на необходимость контроля переобучения. Поэтому в проекте дополнительно сохраняются macro AUROC и macro F1, которые лучше отражают качество на валидационной части.

Кривые обучения на рисунке 5 также помогают принять решение о дальнейших экспериментах. Если после определенного числа эпох метрики перестают улучшаться, можно изменить learning rate, усилить аугментации или ограничить число обучаемых слоев. Эти действия не требуют изменения web-приложения, поскольку относятся только к обучающему контуру.

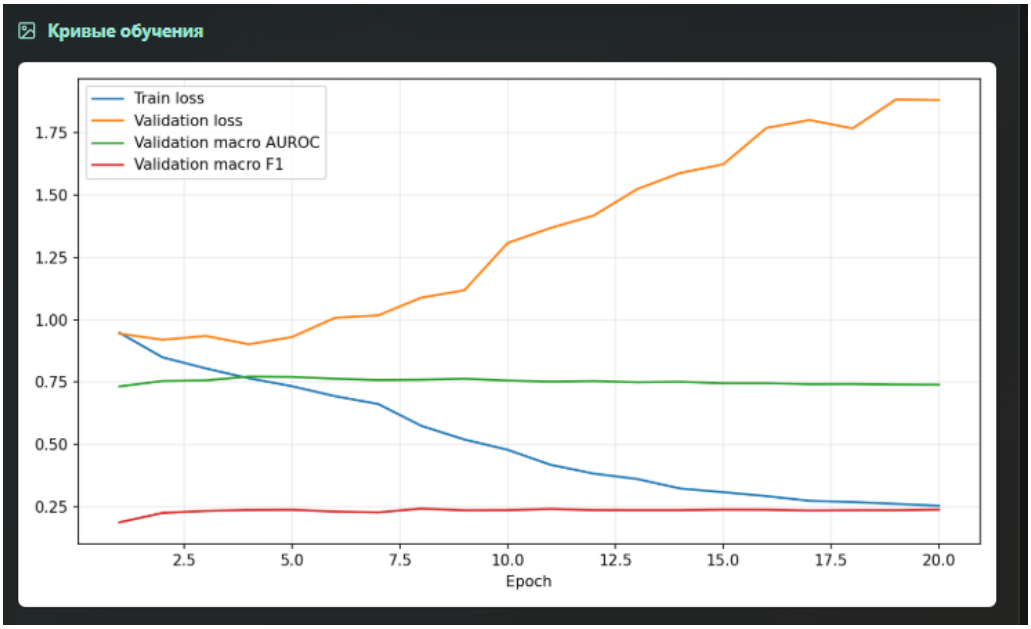


Рисунок 5 - Кривые обучения модели ResNet50

Если validation loss растет слишком быстро, можно остановить обучение раньше и сохранить лучшую эпоху. Такой подход уже заложен в скрипте: веса сохраняются не по последней эпохе, а по лучшему значению validation macro

AUROC. Это уменьшает риск использования переобученной версии модели в пользовательском интерфейсе.

Сохранение curves.png полезно и для отчета по эксперименту. График показывает не только конечное значение метрик, но и поведение модели на протяжении эпох. По нему можно увидеть, насколько стабильно менялись показатели и не возникали ли резкие скачки, указывающие на проблемы с learning rate или данными.

После расчета метрик сохраняется лучшая версия модели:

```
if comparable_auroc > best_macro_auroc:
    best_macro_auroc = comparable_auroc
    best_metrics = val_metrics
    best_thresholds = thresholds
    torch.save({
        "model_state_dict": model.state_dict(),
        "classes": CLASSES,
        "config": config,
        "metrics": val_metrics,
        "thresholds": thresholds,
    }, args.output_dir / "model_best.pt")
```

Файлы, формируемые при обучении, приведены в таблице 13.

Таблица 13 – Файлы, формируемые при обучении

Файл	Содержание
model_best.pt	лучшие веса ResNet50 и служебные данные эксперимента
thresholds.json	индивидуальные пороги классов, подобранные по validation set
classes.json	список 14 классов ChestX-ray14
config.json	параметры обучения, размеры выборок, устройство, learning rate
training_log.csv	история train loss, validation loss, AUROC, F1 и learning rate
metrics_val.json	лучшие метрики на validation split
curves.png	график изменения loss и метрик по эпохам

5.5 Оценка качества модели

Оценка качества вынесена в скрипт app.ml.evaluate. Он загружает обученные веса, применяет модель к выбранному split и сохраняет predictions.csv. В этом файле для каждого снимка фиксируются истинные метки, вероятности, применяемые пороги и бинарные предсказания. На основе

этих данных рассчитываются метрики по каждому классу и усредненные показатели.

В разделе оценки качества запуск скрипта выполняется после завершения обучения и сохранения файла model_best.pt. В скрипт передаются подготовленная таблица с разметкой, каталог изображений, путь к весам и каталог, куда будут сохранены результаты проверки. Такой порядок отделяет обучение от оценки и позволяет повторять проверку модели без повторного обучения.

В реализации рассчитываются AUROC, precision, recall, F1, sensitivity, specificity, accuracy и матрица ошибок для каждого класса. Эти показатели не вынесены в отдельные формулы, поскольку в приложении они вычисляются программно в модуле metrics.py. Если для отдельного класса невозможно посчитать AUROC из-за отсутствия положительных или отрицательных примеров, значение сохраняется как пустое. Это лучше, чем искусственно подставлять нулевое или завышенное значение.

Результаты оценки выводятся как в виде файлов, так и в интерфейсе. Таблица по классам помогает увидеть, что качество модели неодинаково для разных наблюдений. Более высокие значения AUROC получены для Hernia, Effusion, Cardiomegaly и Edema, тогда как для Nodule и Fibrosis результат ниже. Это согласуется с тем, что визуальная выраженность и количество примеров по классам различаются.

Для практического анализа важна не только итоговая средняя метрика. Например, высокий AUROC при низком F1 может означать, что модель в целом умеет ранжировать примеры, но выбранный порог пока не дает хорошего баланса между ложными и пропущенными срабатываниями. Поэтому в интерфейсе отдельно отображаются AUROC, F1, sensitivity, specificity и порог для каждого класса.

Таблица метрик по классам показывает, что значения sensitivity и specificity могут различаться значительно. Высокая specificity означает, что модель редко срабатывает на отрицательных примерах, но этого недостаточно

для оценки полезности: при низкой sensitivity модель может пропускать положительные случаи.

Отдельная строка TP/FP/FN/TN помогает перейти от абстрактной метрики к реальному числу ошибок. Например, высокая доля FN показывает, что модель пропускает положительные примеры, а высокая доля FP говорит о склонности к ложным срабатываниям. Такая детализация полезна при выборе направления дальнейшей настройки.

Порог класса в таблице также важен для объяснения результата пользователю. Если вероятность класса ниже порога, он не попадает в список выбранных меток, даже если входит в top-5. Это поведение отличается от обычной сортировки по вероятности и лучше соответствует многометочной постановке задачи.

Поэтому результаты рассматриваются вместе. AUROC показывает способность разделять примеры при разных порогах, F1 отражает баланс на выбранном пороге, а TP/FP/FN/TN помогает понять, какие ошибки встречаются чаще. Такая интерпретация полезнее, чем простое указание одного среднего числа.

Метрики качества по классам ChestX-ray14 показаны на рисунке 6.

Классы ChestX-ray14

Класс	AUROC	Thr.	F1	Sens.	Spec.	TP/FP/FN/TN
Atelectasis	0.718	0.180	0.291	0.362	0.881	144/447/254/3302
Cardiomegaly	0.838	0.260	0.195	0.169	0.988	15/50/74/4008
Effusion	0.839	0.250	0.448	0.487	0.918	214/303/225/3405
Infiltration	0.734	0.180	0.419	0.607	0.752	406/863/263/2615
Mass	0.723	0.160	0.158	0.124	0.978	24/86/169/3868
Nodule	0.653	0.110	0.180	0.179	0.944	48/217/220/3662
Pneumonia	0.711	0.050	0.105	0.122	0.986	6/59/43/4039
Pneumothorax	0.736	0.070	0.169	0.420	0.873	60/508/83/3496
Consolidation	0.761	0.120	0.190	0.224	0.940	47/238/163/3699
Edema	0.835	0.150	0.187	0.313	0.966	21/137/46/3943
Emphysema	0.808	0.050	0.178	0.267	0.975	16/104/44/3983
Fibrosis	0.694	0.100	0.139	0.173	0.956	23/175/110/3839
Pleural_Thickening	0.700	0.080	0.138	0.359	0.877	46/494/82/3525
Hernia	0.906	0.050	0.000	0.000	1.000	0/0/1/4146

Рисунок 6 - Метрики качества по классам ChestX-ray14

Выходные файлы оценки модели приведены в таблице 14.

Таблица 14 – Выходные файлы оценки модели

Файл	Назначение
metrics.json	полный набор метрик в структурированном виде
metrics.csv	табличное представление метрик по классам
predictions.csv	вероятности, истинные метки, пороги и предсказания по снимкам
roc_curves.png	ROC-кривые по классам
confusion_matrices.png	матрицы ошибок для выбранных порогов

5.6 Анализ ошибок

Отдельный скрипт app.ml.error_analysis предназначен для разбора ошибок после оценки модели. Он читает predictions.csv, выделяет false positive и false negative примеры по каждому классу и сохраняет наиболее заметные случаи. Для таких изображений дополнительно строится Grad-CAM, чтобы было видно, на какую область снимка ориентировалась модель.

Такой анализ нужен не только для демонстрации результата. Он помогает понять, какие классы чаще всего путаются, какие изображения оказываются сложными для модели и не связаны ли ошибки с техническими признаками изображения. Например, модель может уделять внимание краю снимка, буквенным меткам или областям, не относящимся к легочным полям. В таких случаях требуется пересмотреть предобработку или подготовку обучающей выборки.

В процессе анализа ошибок отдельно рассматриваются false positive и false negative. Ложноположительные случаи показывают, где модель видит признак на снимке без соответствующей метки. Ложноотрицательные случаи, наоборот, показывают снимки, где признак указан в разметке, но модель не набрала достаточную вероятность.

Такая проверка полезна для последующего улучшения системы. Если ошибки одного класса связаны с небольшим числом обучающих примеров, нужна балансировка данных или изменение порога. Если Grad-CAM показывает нерелевантные области, нужно пересмотреть предварительную обработку или очистку изображений от технических элементов.

В отчете анализа ошибок сохраняются не только названия файлов, но и вероятности классов. Это помогает отсортировать случаи по уверенности модели и отдельно изучить самые спорные изображения. Например, ложноположительные примеры с высокой вероятностью могут указывать на систематический источник ошибки.

Связь анализа ошибок с Grad-CAM помогает отличить два типа проблем. Если тепловая карта находится в релевантной области, возможно, ошибка связана с шумной разметкой. Если же активация сосредоточена на маркере или рамке снимка, проблема может быть вызвана особенностями данных или предварительной обработки.

При разборе ошибок удобно рассматривать не только отдельные неудачные изображения, но и повторяющиеся закономерности. Если большинство ошибок связано с одним классом, это может указывать на недостаток примеров или пересечение визуальных признаков с другим классом. Если ошибки появляются на снимках с техническими маркерами, нужна дополнительная очистка или маскирование таких элементов.

Результаты анализа ошибок можно использовать как основу для следующего цикла обучения. После выявления проблемных классов можно изменить веса функции потерь, увеличить количество примеров, скорректировать пороги или добавить более подходящие аугментации. Поэтому этот этап не является вспомогательным, а частью общего процесса улучшения модели.

После формирования predictions.csv запускается разбор ошибок. Скрипт получает таблицу предсказаний, находит ложноположительные и ложноотрицательные примеры по каждому классу, сохраняет их в отдельные файлы и при наличии модели строит Grad-CAM для наиболее показательных случаев.

5.7 Реализация Backend API

Backend реализован на FastAPI. При запуске приложения подключаются маршруты классификации, отчетов и метрик обучения. Основной маршрут

POST /api/v1/classify принимает файл, проверяет расширение, ограничивает размер загрузки, выполняет предобработку, запускает модель, строит Grad-CAM и сохраняет отчет в базе данных.

Основной маршрут классификации изображения объединяет загрузку файла, инференс и сохранение результата:

```
@router.post("", response_model=ClassificationResponse)
async def classify_image(file: UploadFile = File(...)):
    content = await file.read()
    if len(content) > MAX_UPLOAD_BYTES:
        raise HTTPException(status_code=413, detail="File is
too large")

    tensor, display_image = load_image(file.filename,
content)
    report_id = str(uuid4())
    probabilities, labels, probs_tensor, top =
model_service.predict(tensor)
    target_index = int(probs_tensor.argmax().item())
    gradcam_service.create_overlay(
        model=model_service.model,
        input_tensor=tensor.to(model_service.device),
        original_image=display_image,
        target_index=target_index,
        output_path=gradcam_path,
    )
```

В ответ клиент получает report_id, название и версию модели, список выбранных меток, вероятности классов, пороги, ссылки на исходное изображение и Grad-CAM, а также предупреждение о назначении результата. Если медицинские веса не загружены и приложение работает в демонстрационном режиме, это также передается в ответе.

Маршрут классификации не возвращает только итоговую метку. В ответ включается полный набор вероятностей по всем классам ChestX-ray14, список top_probabilities, индивидуальные пороги, ссылки на сохраненные изображения и признак demo_mode. Благодаря этому frontend может отобразить как краткий результат, так и расширенную информацию без дополнительных запросов.

В случае ошибки на одном из этапов обработка прерывается до сохранения неполного отчета. Например, если изображение не читается,

запись в SQLite не создается. Если же классификация выполнена, но Grad-CAM по какой-то причине не построен, отчет может быть сохранен с соответствующим сообщением, чтобы пользователь понимал, на каком этапе возникло ограничение.

Основной маршрут классификации формирует отчет только после успешного получения вероятностей. Это исключает появление записей, в которых нет результата модели. При этом изображение и Grad-CAM сохраняются под внутренним идентификатором отчета, поэтому frontend может обращаться к ним через стабильные URL.

В реализации важно, что основной сценарий не зависит от того, открывает пользователь отчет сразу или позже. После сохранения данных все необходимые сведения доступны через отдельные маршруты. Это упрощает frontend: при повторном открытии страницы он получает уже готовый отчет, а не запускает классификацию заново.

Такой подход также уменьшает нагрузку на модель. Повторный просмотр отчета не требует инференса ResNet50 и построения Grad-CAM. Сервер просто возвращает сохраненные метаданные и изображения, что быстрее и стабильнее для пользовательского сценария.

Маршрут отчетов используется не только для истории. Он позволяет повторно открыть результат после перезагрузки страницы и убедиться, что сохраненные вероятности совпадают с тем, что пользователь видел при первичном анализе. Это важно для демонстрации и последующей проверки результатов.

Основные маршруты backend приведены в таблице 15.

Таблица 15 – Основные маршруты backend

Маршрут	Назначение
GET /health	проверка доступности backend
POST /api/v1/classify	загрузка снимка и запуск классификации
GET /api/v1/reports	получение последних отчетов
GET /api/v1/reports/{id}	просмотр сохраненного отчета
GET /api/v1/reports/{id}/image	получение сохраненного изображения

Продолжение таблицы 15.

Маршрут	Назначение
GET /api/v1/reports/{id}/gradcam	получение Grad-CAM overlay
GET /api/v1/training/metrics	получение метрик активного эксперимента
GET /api/v1/training/curves	получение графика обучения

5.8 Хранение отчетов

История анализов хранится в SQLite. Для каждого запроса создается запись с уникальным идентификатором, датой создания, исходным именем файла, безопасным именем сохраненного PNG, версией модели, вероятностями, выбранными метками, путем к исходному изображению и путем к Grad-CAM. Вероятности и метки сохраняются в JSON-формате, так как количество классов фиксировано, но удобнее хранить их как структурированные данные.

При создании отчета сохраняются только технические сведения об анализе:

```
report = create_report(
    report_id=report_id,
    original_filename=Path(file.filename).name,
    safe_filename=safe_filename,
    model_version=model_service.version,
    probabilities=probabilities,
    labels=labels,
    image_path=image_path,
    gradcam_path=gradcam_path,
    warning=model_service.warning,
    demo_mode=not model_service.loaded_medical_weights,
)
```

Сами изображения не помещаются в базу данных. Они сохраняются в файловой системе, а в SQLite хранится только относительный путь. Это упрощает работу с крупными файлами и позволяет при необходимости перенести хранение изображений в отдельное объектное хранилище.

При таком подходе база данных остается небольшой и содержит только структурированную информацию. Это упрощает просмотр истории, фильтрацию отчетов и последующее расширение таблицы. Если понадобится

добавить экспорт результата или хранение дополнительных метрик, структуру отчета можно расширить без изменения формата изображений.

Для безопасности исходное имя файла не используется напрямую как имя в хранилище. Вместо этого формируется безопасное имя или идентификатор отчета. Такой подход снижает риск ошибок из-за специальных символов в имени файла и помогает отделить пользовательское название от внутренней структуры хранения.

Основные поля отчета приведены в таблице 16.

Таблица 16 – Основные поля отчета

Поле	Содержание
id	уникальный идентификатор анализа
created_at	дата и время формирования отчета
original_filename	исходное имя файла без опасных путей
model_version	версия активной модели
probabilities_json	вероятности 14 классов
labels_json	классы, прошедшие порог
image_path	путь к сохраненному PNG
gradcam_path	путь к Grad-CAM overlay
demo_mode	признак работы без обученных медицинских весов

5.9 Реализация Grad-CAM в приложении

Grad-CAM используется в двух местах: при пользовательском анализе снимка и при разборе ошибок после оценки модели. В пользовательском сценарии целевой класс выбирается как класс с максимальной вероятностью. Для этого класса строится тепловая карта, которая накладывается на исходное изображение и сохраняется в формате PNG.

Карта Grad-CAM строится по выбранному классу на основе градиентов последнего сверточного блока:

```
logits = self.model(input_tensor)
score = logits[:, target_index].sum()
score.backward()

weights = gradients[-1].mean(dim=(2, 3), keepdim=True)
cam = (weights * activations[-1]).sum(dim=1, keepdim=True)
cam = F.relu(cam)
cam = F.interpolate(cam, size=(IMAGE_SIZE, IMAGE_SIZE),
mode="bilinear")
```

```
heatmap = cam[0, 0].detach().cpu().numpy()
```

В интерфейсе Grad-CAM отображается рядом с вероятностями. Такая визуализация не считается медицинской разметкой. Ее задача - показать поведение модели и помочь проверить, не опирается ли нейронная сеть на нерелевантные области изображения.

Пример визуального пояснения результата Grad-CAM показан на рисунке 7.

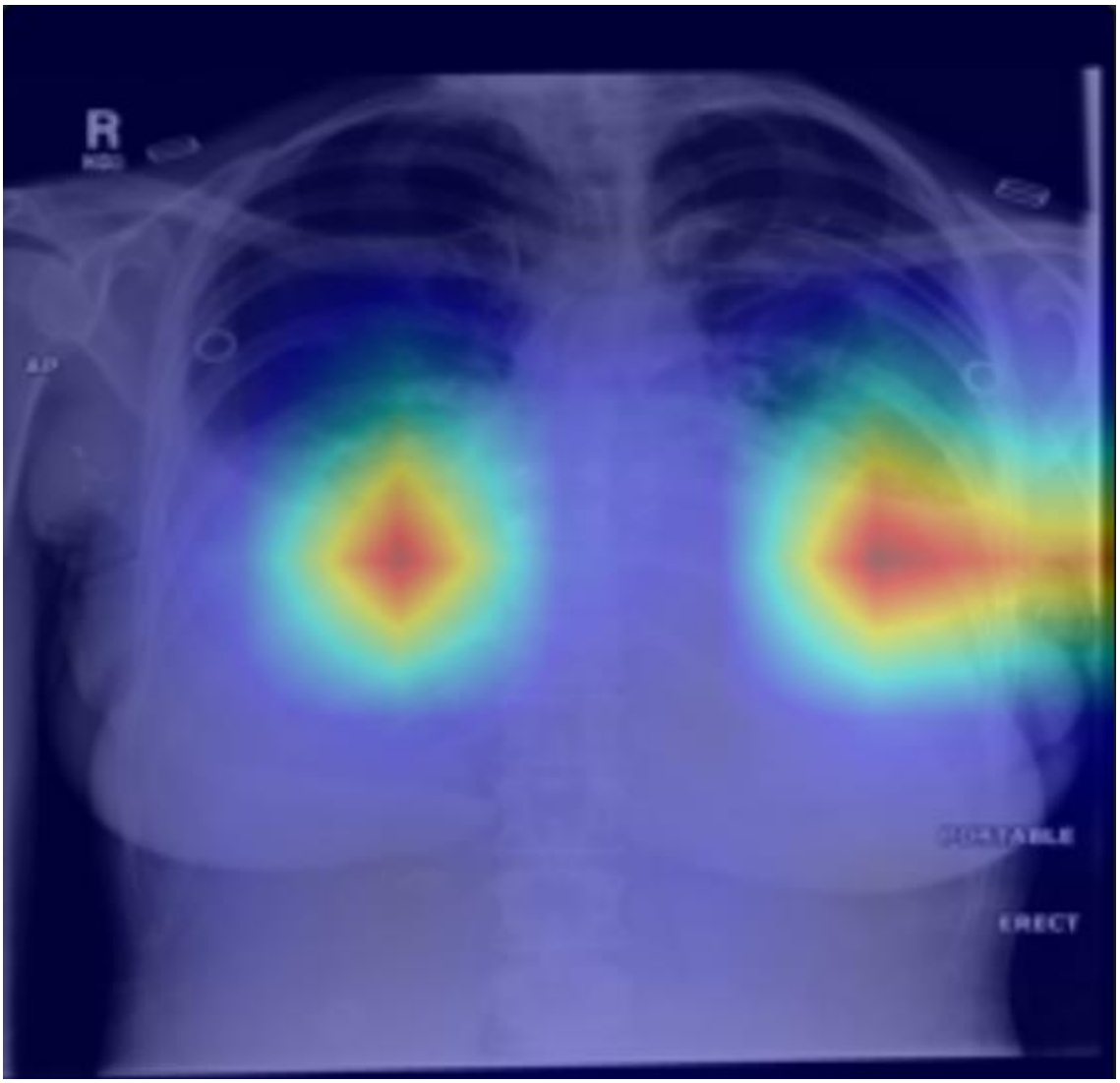


Рисунок 7 – Пример визуального пояснения результата Grad-CAM

5.10 Реализация клиентской части

Клиентская часть реализована на React и TypeScript. Основная страница содержит форму загрузки снимка, кнопку запуска анализа, область результата и боковую панель истории. После выбора файла пользователь запускает

анализ, frontend отправляет файл на backend и после получения ответа показывает карточку результата.

На клиентской стороне результат преобразуется в карточку с вероятностями классов:

```
const probabilities = Object.entries(result.probabilities)
  .sort((a, b) => b[1] - a[1]);

const topProbabilities = result.top_probabilities.length > 0
  ? result.top_probabilities
  : probabilities.map(([label, probability]) => ({
      label,
      probability,
      threshold: result.thresholds?.[label] ?? 0.5,
      selected: result.labels.includes(label),
    }));
```

В карточке результата отображаются исходное изображение, Grad-CAM overlay, выбранные labels, top-5 вероятностей и индивидуальные пороги классов. Такой формат выбран из-за особенностей медицинской классификации: даже если класс не прошел порог, его вероятность может быть близка к решающему значению и полезна для анализа поведения модели.

Экран результата построен так, чтобы пользователь сначала видел наиболее вероятные классы, а затем мог просмотреть полный список значений. Для каждого класса отображается не только процент, но и индивидуальный порог. Это важно, потому что выбранная метка зависит не от места класса в рейтинге, а от сравнения вероятности с порогом.

Отдельный блок с предупреждением расположен в нижней части результата. Он напоминает, что вывод модели не является медицинским заключением. Такое предупреждение оставлено в интерфейсе постоянно, а не только на стартовой странице, потому что пользователь может открыть отчет напрямую из истории.

Отображение вероятностей классов в интерфейсе показано на рисунке 8.

Рисунок 8 – Отображение вероятностей классов в интерфейсе

5.11 Запуск приложения

Приложение можно запустить двумя способами: через Docker Compose или локально, отдельно для backend и frontend. Docker Compose удобен для демонстрации готового приложения, но обучение модели имеет смысл запускать в локальном CUDA-окружении. Это связано с тем, что стандартный backend-контейнер в проекте не настроен как специализированная среда для обучения на GPU.

При локальном запуске backend поднимается командой uvicorn, а frontend - через npm run dev. После запуска frontend доступен на localhost:5173, backend - на localhost:8000. Backend автоматически ищет веса модели и thresholds.json в каталоге активного эксперимента. Пути можно переопределить через переменные окружения.

6 Тестирование и проверка работы системы

Проверка системы проводилась по двум направлениям. Первое направление связано с машинным обучением: подготовка датасета, обучение модели, расчет метрик, подбор порогов и анализ ошибок. Второе направление относится к web-приложению: загрузка файлов, работа API, сохранение отчетов, отображение истории и Grad-CAM в интерфейсе.

6.1 Проверка подготовки данных

На этапе подготовки проверялось, что скрипт корректно читает Data_Entry_2017.csv, разбирает Finding Labels, формирует бинарные колонки классов и отбрасывает записи без доступного изображения. Дополнительно проверялось, что данные разделяются на train, validation и test без смешивания снимков одного пациента между разными частями выборки при наличии patient_id.

Отдельно проверялись случаи, когда изображение отсутствует в каталоге, а запись о нем есть в CSV. Такие строки не должны попадать в обучающую выборку, иначе во время обучения возникнет ошибка чтения файла. После подготовки сверялось количество строк в исходной и итоговой таблице, а также количество примеров по каждому классу.

Проверка split выполнялась по patient_id. Для нескольких случайно выбранных пациентов контролировалось, что все их снимки относятся только к одной из подвыборок. Это уменьшает риск завышения качества и делает итоговую оценку модели более надежной.

После подготовки также проверялось, что каждая строка содержит значения всех четырнадцати классов. Это защищает обучение от ошибок, когда часть колонок отсутствует или имеет строковый формат вместо числового. Наличие такого контроля делает подготовку данных более надежной.

Результаты проверки подготовки данных позволяют убедиться, что дальнейшее обучение начинается с корректной таблицы. Если на этом этапе допустить ошибку, последующие метрики будут вводить в заблуждение, потому что модель будет обучаться на неверно сформированных метках.

Результаты проверки подготовки датасета приведены в таблице 17.

Таблица 17 – Проверка подготовки датасета

Проверка	Ожидаемый результат
Наличие поля Finding Labels	CSV принимается только при наличии разметки
Разбор нескольких меток	строка вида A|B преобразуется в несколько единиц в бинарном векторе
No Finding	строка переводится в нулевой вектор по 14 классам
Отсутствующий PNG	строка исключается, если режим keep-missing-images не включен
Разделение по пациентам	снимки одного patient_id попадают только в один split

6.2 Проверка обучения и оценки

Во время обучения контролировались loss на обучающей выборке, loss на validation split, macro AUROC и macro F1. Если значение macro AUROC улучшалось, веса модели сохранялись как model_best.pt. После обучения выполнялась оценка на test split, где формировались predictions.csv, metrics.json, metrics.csv, ROC-кривые и матрицы ошибок.

Важная часть проверки - отсутствие фиктивных результатов. Метрики появляются только после запуска evaluate.py на реальном подготовленном датасете. Если веса не обучены или файл модели отсутствует, приложение не должно выдавать демонстрационные значения как результат реального медицинского анализа.

Экран метрик обучения дополнительно использовался для визуальной проверки результатов эксперимента. На нем отображаются macro AUROC, micro AUROC, macro F1 и micro F1, а также таблица по всем классам ChestX-ray14. Такой вид удобен для сравнения качества по отдельным наблюдениям без открытия JSON-файлов вручную.

Экран просмотра метрик обучения показан на рисунке 9.

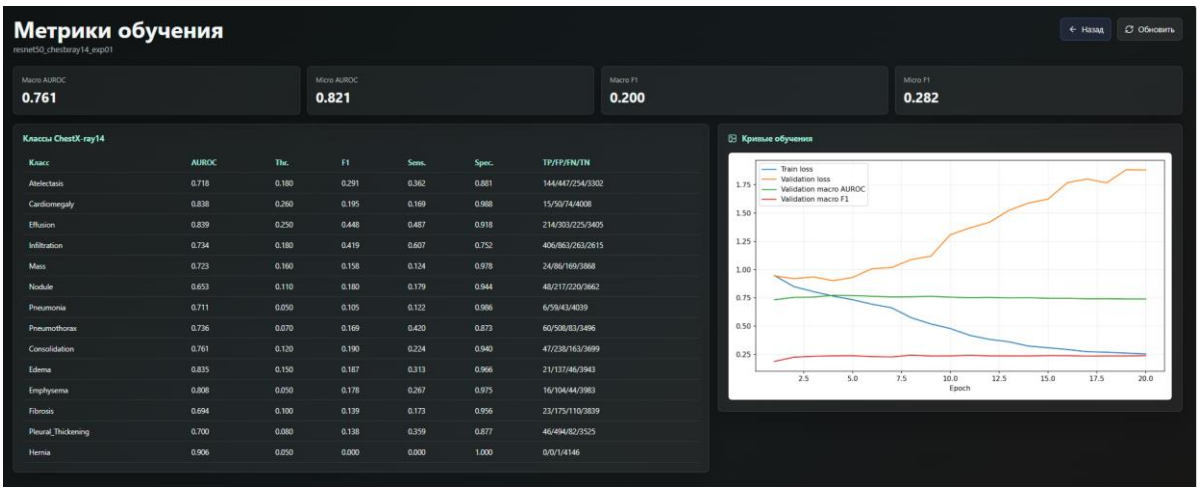


Рисунок 9 – Экран просмотра метрик обучения

На рисунке 9 видно, что итоговая оценка складывается из нескольких показателей. Macro AUROC характеризует среднее качество разделения по классам, micro AUROC учитывает все примеры вместе, а F1 показывает баланс между полнотой и точностью при выбранных порогах. Таблица по классам помогает быстро найти наблюдения, по которым модель требует дополнительной настройки.

По полученным значениям видно, что модель лучше разделяет некоторые классы по AUROC, чем по F1. Это означает, что модель в целом формирует осмысленный порядок вероятностей, но выбранные пороги еще требуют настройки. Поэтому результаты обучения рассматриваются как основа для дальнейшего улучшения модели, а не как окончательная медицинская оценка.

6.3 Проверка API и обработки ошибок

При проверке backend API анализировались типовые сценарии: загрузка корректного изображения, загрузка DICOM-файла, обращение к истории отчетов, получение изображения Grad-CAM и обработка некорректного файла. Для каждого сценария проверялось, что API возвращает понятный ответ и не создает отчет при ошибке чтения входных данных.

Проверка API выполнялась не только по успешному сценарию. Для неподдерживаемого расширения ожидался отказ до чтения изображения, для поврежденного файла - ошибка чтения, для слишком большого файла -

ограничение размера загрузки. Такая проверка нужна, чтобы пользователь не получал необработанные внутренние исключения.

Также проверялось, что после успешной классификации создается отчет, а ссылки на исходное изображение и Grad-CAM открываются через отдельные маршруты. Это подтверждает связность backend-части: результат анализа не теряется после ответа API и доступен при повторном открытии отчета.

Дополнительно проверялось соответствие формата ответа API ожиданиям frontend. Если backend возвращает вероятности, пороги или ссылки в измененном виде, клиентская часть может не отобразить результат корректно. Поэтому структура JSON-ответа фиксируется в схемах и используется при разработке интерфейса.

Проверка ошибок выполнялась вручную и через типовые запросы. В каждом случае контролировалось не только наличие кода ошибки, но и понятный текст сообщения. Это важно, потому что пользователь не должен разбираться в деталях Python-исключений или внутренней структуре сервера.

Для успешного сценария контролировалась целостность ответа: наличие report_id, model_version, probabilities, labels, thresholds и ссылок на изображения. Если хотя бы одно из этих полей отсутствует, frontend не сможет корректно собрать карточку результата. Поэтому проверка API связана не только с backend, но и с работой клиентской части.

Для сценария с DICOM дополнительно проверялось, что в отчет не попадают персональные поля. В ответе и сохраненной записи используются только безопасные технические данные. Это соответствует требованию минимизировать обработку лишних медицинских сведений.

Проверка API приведена в таблице 18.

Таблица 18 – Проверка API

Сценарий	Ожидаемое поведение
POST /api/v1/classify с PNG	возвращается report_id, probabilities, labels, gradcam_url
POST /api/v1/classify с DICOM	читаются пиксельные данные, персональные поля не сохраняются

Продолжение таблицы 18.

Сценарий	Ожидаемое поведение
Загрузка неподдерживаемого файла	возвращается ошибка 415
Поврежденное изображение	возвращается ошибка 400
GET /api/v1/reports/{id}	возвращается сохраненный отчет
GET /api/v1/training/metrics	возвращаются метрики активного эксперимента или предупреждение об их отсутствии

6.4 Проверка пользовательского интерфейса

Клиентскую часть проверяли в сценарии, близком к обычной работе пользователя. Пользователь выбирает файл, запускает анализ и получает результат через интерфейс, без перехода к внутренним маршрутам API. После успешной классификации приложение открывает страницу /reports/{report_id}, а новый отчет появляется в истории.

Интерфейс показывает предупреждение о назначении результата. Это нужно, чтобы пользователь сразу видел ограничение системы: приложение создано для исследования и демонстрации работы модели, а не для самостоятельной постановки медицинского заключения.

При проверке смотрели не только загрузку изображения, но и вывод полного списка классов. В ChestX-ray14 используется 14 наблюдений, поэтому таблица вероятностей должна оставаться читаемой даже в небольшом окне браузера. Для этого классы выводятся отдельными строками, рядом показываются прогресс-бары и числовые значения.

При открытии сохраненного отчета проверяли, что система показывает тот же набор меток, ту же версию модели и то же изображение Grad-CAM. Для воспроизводимости это принципиально: пользователь должен видеть не результат нового инференса, а отчет, который система сформировала при первой обработке снимка.

Проверка интерфейса включала обновление страницы и возврат к списку отчетов. В таких сценариях приложение повторно запрашивает данные у

backend и не теряет состояние анализа. Это подтверждает, что результат хранится на сервере, а не только в памяти браузера.

Grad-CAM отображался вместе с исходным снимком. Пользователь должен видеть, какие области активации соответствуют реальному изображению. Если оставить только тепловую карту без исходного снимка, интерпретировать результат становится сложнее.

Отдельно проверяли экран метрик обучения. Он нужен не рядовому пользователю, а разработчику или исследователю, который оценивает текущую версию модели. На одном экране собраны усредненные показатели, таблица по классам и кривые обучения. Не приходится вручную открывать несколько файлов эксперимента.

Также проверяли, как таблица вероятностей и карточка результата выглядят в темной теме. Для исследовательского интерфейса важно, чтобы числовые значения, названия классов и предупреждение читались без дополнительного масштабирования страницы.

6.5 Анализ ограничений

У итоговой системы есть ограничения, даже при наличии обучения, метрик и web-интерфейса. ChestX-ray14 содержит шумную разметку: метки в нем получены из текстовых медицинских отчетов, а такие данные не всегда бывают точными. Качество модели зависит от того, насколько полно скачан датасет и какие параметры использовались при обучении. Grad-CAM помогает увидеть, куда смотрит модель, но не подтверждает медицинскую правильность ее вывода.

Изменение preprocessing тоже влияет на результат. Если добавить обрезку области легких, center crop или отдельное выравнивание снимков, распределение входных данных изменится. В этом случае ранее подобранные thresholds могут перестать подходить, поэтому модель придется дообучать или обучать заново.

Есть и ограничение, связанное с источником данных. Модель обучалась на конкретном наборе снимков. При работе с изображениями из другой

клиники или другого архива могут измениться контрастность, разрешение, положение пациента, формат снимка и наличие технических маркеров. Перед практическим использованием систему нужно проверять на независимом наборе изображений.

Grad-CAM показывает поведение модели, но не является точной медицинской сегментацией. Если тепловая карта попадает в область, которая действительно интересна при анализе снимка, это повышает доверие к результату. Но сама по себе такая карта не доказывает, что класс выбран правильно. Поэтому визуальное пояснение нужно рассматривать вместе с метриками и анализом ошибок.

Эти ограничения не уменьшают ценность разработанного комплекса, но задают границы его применения. Система показывает, как в одном приложении можно объединить обучение модели, расчет метрик, пользовательский анализ и визуальное пояснение результата. Для перехода к реальному медицинскому использованию потребуются отдельная проверка, независимая оценка качества и другой уровень ответственности.

По этой причине в интерфейсе и текстах отчета указано, что результат имеет исследовательский характер. Это не формальная надпись. Пользователь должен понимать: вероятность класса и тепловая карта показывают работу модели, а не готовое заключение врача.

6.6 Вывод по тестированию

Проверка подтвердила, что основные контуры системы работают: подготовка ChestX-ray14, обучение ResNet50, расчет метрик, анализ ошибок, загрузка пользовательского снимка, построение Grad-CAM и сохранение отчета. Реализованный комплекс можно использовать для дальнейших экспериментов с качеством модели и для демонстрации результата классификации через web-интерфейс.

Проверка также показала, что части приложения связаны в единый маршрут. Подготовленный датасет поступает в обучение, backend-сервис загружает веса модели, результат инференса сохраняется в отчет, а frontend

выводит сохраненные изображения и метрики. Это подтверждает работу не только отдельных функций, но и всего программного комплекса.

При тестировании стали видны направления для доработки. В первую очередь нужно повышать качество модели по редким классам, точнее подбирать thresholds, расширять набор тестовых снимков и вручную анализировать Grad-CAM-карты. Эти шаги не требуют менять архитектуру системы, но могут заметно улучшить итоговое качество решения.

7 Итоговый результат разработки

Итогом разработки стал программный комплекс MedVision CNN. Он объединяет инструменты подготовки медицинского датасета, обучение ResNet50, оценку качества, анализ ошибок и web-приложение для пользовательского инференса. Такое объединение удобно для исследования: результаты обучения можно сразу использовать в backend, а затем проверить работу модели через интерфейс.

Основной практический результат состоит в том, что был реализован полный маршрут обработки снимка: загрузка файла, приведение изображения к нужному формату, получение вероятностей по классам ChestX-ray14, применение индивидуальных порогов, построение Grad-CAM и сохранение отчета. При наличии обученных весов система использует их для инференса; при отсутствии весов приложение должно явно показывать демонстрационный режим.

Разработанная структура оставляет возможность дальнейшего развития. В систему можно добавить другие архитектуры, расширить preprocessing, подключить PostgreSQL, добавить экспорт отчета в PDF и провести более строгую проверку качества на независимой выборке. При этом текущая версия уже содержит основные компоненты, необходимые для исследовательской работы с рентгеновскими снимками грудной клетки.

ЗАКЛЮЧЕНИЕ

В ходе разработки реализована web-система MedVision CNN для автоматической классификации рентгеновских снимков грудной клетки. Серверная часть построена на FastAPI и PyTorch, клиентская часть - на React и TypeScript. Для классификации используется ResNet50, обучаемая на подготовленном ChestX-ray14 в многометочной постановке.

В системе реализованы подготовка метаданных датасета, разделение данных на train, validation и test, обучение модели с учетом дисбаланса классов, подбор индивидуальных порогов, расчет метрик качества, анализ ошибок и построение Grad-CAM. Для пользовательского режима предусмотрены загрузка PNG, JPG и DICOM, сохранение истории отчетов в SQLite и отображение результата в web-интерфейсе.

Полученный результат не относится к медицинским диагностическим продуктам. Он нужен для исследования и демонстрации работы сверточной нейронной сети в задаче классификации медицинских снимков. Для применения в клиническом процессе потребуются дополнительная валидация, экспертная проверка, контроль качества данных и оценка устойчивости модели на независимых выборках.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Wang X., Peng Y., Lu L., Lu Z., Bagheri M., Summers R. M. ChestX-ray8: Hospital-scale Chest X-ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. – 2017. – URL: https://arxiv.org/abs/1705.02315 (дата обращения: 25.04.2025).

2. Selvaraju R. R., Cogswell M., Das A., Vedantam R., Parikh D., Batra D. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization // IEEE International Conference on Computer Vision. – 2017. – URL: https://openaccess.thecvf.com/content_iccv_2017/html/Selvaraju_Grad-CAM_Visual_Explanations_ICCV_2017_paper.html (дата обращения: 25.04.2025).

3. Goodfellow I., Bengio Y., Courville A. Deep Learning. – Cambridge: MIT Press, 2016. – URL: https://www.deeplearningbook.org/ (дата обращения: 24.04.2025).

4. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. – 2016. – URL: https://arxiv.org/abs/1512.03385 (дата обращения: 24.04.2025).

5. Paszke A., Gross S., Massa F., Lerer A., Bradbury J., Chanan G., Killeen T., Lin Z. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library // Advances in Neural Information Processing Systems. – 2019. – Vol. 32. – URL: https://arxiv.org/abs/1912.01703 (дата обращения: 24.04.2025).

6. Rajpurkar P., Irvin J., Zhu K., Yang B., Mehta H., Duan T., Ding D., Bagul A. et al. CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning. – 2017. – URL: https://arxiv.org/abs/1711.05225 (дата обращения: 24.04.2025).

7. Cohen J. P., Viviano J. D., Bertin P., Morrison P., Torabian P., Guarrera M., Lungren M. P., Chaudhari A. et al. TorchXRayVision: A library of chest X-ray

datasets and models. – 2021. – URL: https://arxiv.org/abs/2111.00595 (дата обращения: 24.04.2025).

8. Cardoso M. J., Li W., Brown R., Ma N., Kerfoot E., Wang Y., Murrey B., Myronenko A. et al. MONAI: An open-source framework for deep learning in healthcare. – 2022. – URL: https://arxiv.org/abs/2211.02701 (дата обращения: 24.04.2025).

9. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures: Doctoral dissertation. – University of California, Irvine, 2000. – URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (дата обращения: 24.04.2025).

10. FastAPI. Request Files [Электронный ресурс]. – URL: https://fastapi.tiangolo.com/tutorial/request-files/ (дата обращения: 25.05.2026).

11. pydicom. Working with Pixel Data [Электронный ресурс]. – URL: https://pydicom.github.io/pydicom/stable/guides/user/working_with_pixel_data.html (дата обращения: 24.04.2025).

ПРИЛОЖЕНИЕ А

(обязательное)

Применение сверточных нейронных сетей для автоматической классификации медицинских снимков

Листинг программы

Файл gradcam_service.py

```
from __future__ import annotations
from pathlib import Path
import torch
from fastapi import HTTPException
from PIL import Image
from app.ml.gradcam import save_gradcam_overlay
class GradCamService:
    def create_overlay(
        self,
        model: torch.nn.Module,
        input_tensor: torch.Tensor,
        original_image: Image.Image,
        target_index: int,
        output_path: Path,
    ) -> None:
        if model is None:
            raise HTTPException(status_code=500, detail="Model
is not loaded")
        try:
            save_gradcam_overlay(
                model=model,
                input_tensor=input_tensor,
                original_image=original_image,
                target_index=target_index,
                output_path=output_path,
            )
        except Exception as exc:
            raise HTTPException(status_code=500, detail=f"Grad-
CAM failed: {exc}") from exc


gradcam_service = GradCamService()
```

Файл model_service.py

```
from __future__ import annotations

import json
import threading
from pathlib import Path

import torch
from fastapi import HTTPException

from app.config import (
    DEMO_WARNING_TEXT,
    LABEL_THRESHOLD,
    LEGACY_WEIGHTS_PATH,
    MODEL_VERSION,
    NO_IMAGENET_WARNING_TEXT,
    THRESHOLDS_PATH,
```

```
    WARNING_TEXT,
    WEIGHTS_PATH,
)
from app.ml.classes import CLASSES
from app.ml.model import load_model_checkpoint


class ModelService:
    def __init__(self) -> None:
        self.model: torch.nn.Module | None = None
        self.loaded_medical_weights = False
        self.loaded_imagenet_weights = False
        self.thresholds = {class_name: LABEL_THRESHOLD for
class_name in CLASSES}
        self.device = torch.device("cuda" if
torch.cuda.is_available() else "cpu")
        self._lock = threading.Lock()

    def load(self) -> None:
        with self._lock:
            if self.model is not None:
                return
            try:
                torch.manual_seed(42)
                weights_path = WEIGHTS_PATH if
WEIGHTS_PATH.exists() else LEGACY_WEIGHTS_PATH
                model, loaded_medical_weights =
load_model_checkpoint(weights_path)
                model.to(self.device)
                model.eval()
                self.thresholds =
self._load_thresholds(weights_path)
            except Exception as exc:
                raise HTTPException(status_code=500,
detail=f"Model failed to load: {exc}") from exc
            self.model = model
            self.loaded_medical_weights = loaded_medical_weights
            self.loaded_imagenet_weights = bool(getattr(model,
"loaded_imagenet_weights", False))

    def _load_thresholds(self, weights_path: Path) -> dict[str,
float]:
        candidates = [
            THRESHOLDS_PATH,
            weights_path.parent / "thresholds.json",
        ]
        for candidate in candidates:
            if candidate.exists():
                data =
json.loads(candidate.read_text(encoding="utf-8"))
                return {
                    class_name: float(data.get(class_name,
LABEL_THRESHOLD))
```

```
                    for class_name in CLASSES
                }
        return {class_name: LABEL_THRESHOLD for class_name in
CLASSES}

    def predict(self, tensor: torch.Tensor) -> tuple[dict[str,
float], list[str], torch.Tensor, list[dict]]:
        if self.model is None:
            self.load()
        if self.model is None:
            raise HTTPException(status_code=500, detail="Model
is not loaded")

        input_tensor = tensor.to(self.device)
        with torch.no_grad():
            logits = self.model(input_tensor)
            probabilities_tensor =
torch.sigmoid(logits).detach().cpu()[0]

        probabilities = {
            class_name:
round(float(probabilities_tensor[index]), 4)
            for index, class_name in enumerate(CLASSES)
        }
        top_probabilities = [
            {
                "label": class_name,
                "probability": probability,
                "threshold":
round(float(self.thresholds.get(class_name, LABEL_THRESHOLD)),
4),
                "selected": probability >=
float(self.thresholds.get(class_name, LABEL_THRESHOLD)),
            }
            for class_name, probability in
sorted(probabilities.items(), key=lambda item: item[1],
reverse=True)
        ]
        labels = [item["label"] for item in top_probabilities if
item["selected"]]
        if not labels:
            labels = [top_probabilities[0]["label"]]

        return probabilities, labels, probabilities_tensor,
top_probabilities

    @property
    def version(self) -> str:
        if self.loaded_medical_weights:
            return MODEL_VERSION
        return f"{MODEL_VERSION}-demo"

    @property
```

```
    def warning(self) -> str:
        if not self.loaded_medical_weights:
            return DEMO_WARNING_TEXT
        if not self.loaded_imagenet_weights:
            return f"{WARNING_TEXT}. {NO_IMAGENET_WARNING_TEXT}"
        return WARNING_TEXT


model_service = ModelService()
```

Файл preprocessing.py

```
from __future__ import annotations

import io
from pathlib import Path

import numpy as np
import torch
from fastapi import HTTPException
from PIL import Image, ImageOps

from app.config import ALLOWED_EXTENSIONS
from app.ml.transforms import inference_transforms

try:
    import pydicom
except Exception:  # pragma: no cover - optional dependency
    pydicom = None


IMAGE_EXTENSIONS = {".png", ".jpg", ".jpeg"}
DICOM_EXTENSIONS = {".dcm", ".dicom"}

_transform = inference_transforms()


def validate_extension(filename: str) -> str:
    extension = Path(filename).suffix.lower()
    if extension not in ALLOWED_EXTENSIONS:
        raise HTTPException(status_code=415, detail="Unsupported
file format")
    return extension


def load_image(filename: str, content: bytes) ->
tuple[torch.Tensor, Image.Image]:
    extension = validate_extension(filename)

    try:
        if extension in IMAGE_EXTENSIONS:
            image = Image.open(io.BytesIO(content))
```

```
            display_image =
ImageOps.exif_transpose(image).convert("L").convert("RGB")
        elif extension in DICOM_EXTENSIONS:
            display_image = _load_dicom_pixels(content)
        else:
            raise HTTPException(status_code=415,
detail="Unsupported file format")
    except HTTPException:
        raise
    except Exception as exc:
        raise HTTPException(status_code=400, detail="File is
corrupted or unreadable") from exc

    tensor = _transform(display_image).unsqueeze(0)
    return tensor, display_image


def _load_dicom_pixels(content: bytes) -> Image.Image:
    if pydicom is None:
        raise HTTPException(status_code=415, detail="DICOM
support requires pydicom")

    # Only pixel data is converted; patient metadata is neither
returned nor saved.
    dataset = pydicom.dcmread(io.BytesIO(content), force=True,
stop_before_pixels=False)
    if "PixelData" not in dataset:
        raise HTTPException(status_code=400, detail="DICOM file
has no pixel data")

    pixels = dataset.pixel_array.astype(np.float32)
    slope = float(getattr(dataset, "RescaleSlope", 1.0))
    intercept = float(getattr(dataset, "RescaleIntercept", 0.0))
    pixels = pixels * slope + intercept

    pixels -= np.min(pixels)
    max_value = float(np.max(pixels))
    if max_value > 0:
        pixels /= max_value
    pixels = (pixels * 255).astype(np.uint8)

    if getattr(dataset, "PhotometricInterpretation", "").upper()
== "MONOCHROME1":
        pixels = 255 - pixels

    return Image.fromarray(pixels).convert("L").convert("RGB")


def save_sanitized_image(image: Image.Image, path: Path) ->
None:
    path.parent.mkdir(parents=True, exist_ok=True)
    image.save(path, format="PNG")
```

Файл report_service.py

```
from __future__ import annotations

import json
from datetime import datetime, timezone
from pathlib import Path

from app.config import API_PREFIX, BASE_DIR, LABEL_THRESHOLD,
MODEL_NAME, THRESHOLDS_PATH
from app.database.db import get_connection, relative_to_backend
from app.ml.classes import CLASSES


def create_report(
    report_id: str,
    original_filename: str,
    safe_filename: str,
    model_version: str,
    probabilities: dict[str, float],
    labels: list[str],
    image_path: Path,
    gradcam_path: Path,
    warning: str,
    demo_mode: bool,
) -> dict:
    created_at = datetime.now(timezone.utc).isoformat()
    with get_connection() as conn:
        conn.execute(
            """
            INSERT INTO reports (
                id, created_at, original_filename,
safe_filename, model_name, model_version,
                probabilities_json, labels_json, image_path,
gradcam_path, warning, demo_mode
            )
            VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
            """,
            (
                report_id,
                created_at,
                original_filename,
                safe_filename,
                MODEL_NAME,
                model_version,
                json.dumps(probabilities, ensure_ascii=False),
                json.dumps(labels, ensure_ascii=False),
                relative_to_backend(image_path),
                relative_to_backend(gradcam_path),
                warning,
                int(demo_mode),
            ),
        )
```

```
    return get_report(report_id)


def list_reports() -> list[dict]:
    with get_connection() as conn:
        rows = conn.execute(
            "SELECT * FROM reports ORDER BY created_at DESC
LIMIT 50"
        ).fetchall()
    return [_row_to_summary(row) for row in rows]


def get_report(report_id: str) -> dict:
    with get_connection() as conn:
        row = conn.execute("SELECT * FROM reports WHERE id = ?",
(report_id,)).fetchone()
    if row is None:
        return {}
    return _row_to_detail(row)


def get_gradcam_path(report_id: str) -> Path | None:
    with get_connection() as conn:
        row = conn.execute(
            "SELECT gradcam_path FROM reports WHERE id = ?",
(report_id,)
        ).fetchone()
    if row is None:
        return None
    return BASE_DIR / row["gradcam_path"]


def get_image_path(report_id: str) -> Path | None:
    with get_connection() as conn:
        row = conn.execute("SELECT image_path FROM reports WHERE
id = ?", (report_id,)).fetchone()
    if row is None:
        return None
    return BASE_DIR / row["image_path"]


def _row_to_summary(row) -> dict:
    labels = json.loads(row["labels_json"])
    return {
        "id": row["id"],
        "created_at": row["created_at"],
        "original_filename": row["original_filename"],
        "model_name": row["model_name"],
        "labels": labels,
        "model_version": row["model_version"],
        "gradcam_url":
f"{API_PREFIX}/reports/{row['id']}/gradcam",
        "warning": row["warning"],
```

```
        "demo_mode": bool(row["demo_mode"]),
    }


def _row_to_detail(row) -> dict:
    summary = _row_to_summary(row)
    probabilities = json.loads(row["probabilities_json"])
    thresholds = _load_thresholds()
    summary.update(
        {
            "probabilities": probabilities,
            "thresholds": thresholds,
            "top_probabilities":
_top_probabilities(probabilities, thresholds),
            "image_url":
f"{API_PREFIX}/reports/{row['id']}/image",
        }
    )
    return summary


def _load_thresholds() -> dict[str, float]:
    if THRESHOLDS_PATH.exists():
        data =
json.loads(THRESHOLDS_PATH.read_text(encoding="utf-8"))
        return {
            class_name: float(data.get(class_name,
LABEL_THRESHOLD))
            for class_name in CLASSES
        }
    return {class_name: LABEL_THRESHOLD for class_name in
CLASSES}


def _top_probabilities(probabilities: dict[str, float],
thresholds: dict[str, float]) -> list[dict]:
    return [
        {
            "label": class_name,
            "probability": probability,
            "threshold": round(float(thresholds.get(class_name,
LABEL_THRESHOLD)), 4),
            "selected": probability >=
float(thresholds.get(class_name, LABEL_THRESHOLD)),
        }
        for class_name, probability in sorted(
            probabilities.items(),
            key=lambda item: item[1],
            reverse=True,
        )
    ]
```

Файл schemas.py

```
from pydantic import BaseModel, Field


class TopProbability(BaseModel):
    label: str
    probability: float
    threshold: float
    selected: bool


class ClassificationResponse(BaseModel):
    report_id: str
    model_name: str = "resnet50"
    model_version: str
    labels: list[str]
    probabilities: dict[str, float]
    thresholds: dict[str, float] = Field(default_factory=dict)
    top_probabilities: list[TopProbability] =
Field(default_factory=list)
    image_url: str
    gradcam_url: str
    warning: str
    demo_mode: bool = False


class ReportSummary(BaseModel):
    id: str
    created_at: str
    original_filename: str
    model_name: str = "resnet50"
    labels: list[str]
    model_version: str
    gradcam_url: str
    warning: str
    demo_mode: bool = False


class ReportDetail(ReportSummary):
    probabilities: dict[str, float] =
Field(default_factory=dict)
    thresholds: dict[str, float] = Field(default_factory=dict)
    top_probabilities: list[TopProbability] =
Field(default_factory=list)
    image_url: str


class TrainingMetricsResponse(BaseModel):
    experiment: str | None = None
    metrics: dict = Field(default_factory=dict)
    classes: list[str] = Field(default_factory=list)
    curves_url: str | None = None
```

```
    warning: str | None = None
```

Файл train.py

```
from __future__ import annotations

import argparse
import json
from pathlib import Path

import matplotlib

matplotlib.use("Agg")

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import torch
from torch import nn
from torch.utils.data import DataLoader

from app.config import LABEL_THRESHOLD
from app.ml.classes import CLASSES
from app.ml.dataset import ChestXray14Dataset
from app.ml.metrics import compute_multilabel_metrics,
optimize_thresholds
from app.ml.model import build_resnet50
from app.ml.transforms import eval_transforms, train_transforms


def parse_args() -> argparse.Namespace:
    parser = argparse.ArgumentParser(description="Train ResNet50
on prepared ChestX-ray14 CSV.")
    parser.add_argument("--csv", required=True, type=Path)
    parser.add_argument("--images-root", required=True,
type=Path)
    parser.add_argument("--epochs", type=int, default=20)
    parser.add_argument("--batch-size", type=int, default=16)
    parser.add_argument("--lr", type=float, default=1e-4)
    parser.add_argument("--output-dir", required=True,
type=Path)
    parser.add_argument("--num-workers", type=int, default=0)
    parser.add_argument("--threshold", type=float,
default=LABEL_THRESHOLD)
    parser.add_argument("--seed", type=int, default=42)
    parser.add_argument("--no-pretrained", action="store_true")
    parser.add_argument("--no-pos-weight", action="store_true")
    parser.add_argument("--limit", type=int, default=0,
help="Optional row limit for smoke tests.")
    return parser.parse_args()
```

```
def maybe_limit(dataset: ChestXray14Dataset, limit: int) ->
ChestXray14Dataset:
    if limit > 0:
        dataset.dataframe =
dataset.dataframe.head(limit).reset_index(drop=True)
    return dataset


def compute_pos_weight(dataset: ChestXray14Dataset, device:
torch.device) -> torch.Tensor:
    positives =
dataset.dataframe[CLASSES].sum(axis=0).astype("float32").to_nump
y()
    negatives = len(dataset) - positives
    weights = np.divide(
        negatives,
        np.maximum(positives, 1.0),
        out=np.ones_like(positives, dtype=np.float32),
        where=positives > 0,
    )
    weights = np.clip(weights, 1.0, 25.0)
    return torch.tensor(weights, dtype=torch.float32,
device=device)


def run_train_epoch(
    model: nn.Module,
    loader: DataLoader,
    criterion: nn.Module,
    optimizer: torch.optim.Optimizer,
    device: torch.device,
) -> float:
    model.train()
    total_loss = 0.0
    total_examples = 0
    for images, labels in loader:
        images = images.to(device)
        labels = labels.to(device)
        optimizer.zero_grad(set_to_none=True)
        logits = model(images)
        loss = criterion(logits, labels)
        loss.backward()
        optimizer.step()
        total_loss += float(loss.item()) * images.size(0)
        total_examples += images.size(0)
    return total_loss / max(total_examples, 1)


@torch.no_grad()
def evaluate_loader(
    model: nn.Module,
    loader: DataLoader,
    criterion: nn.Module,
```

```
    device: torch.device,
) -> tuple[float, np.ndarray, np.ndarray]:
    model.eval()
    total_loss = 0.0
    total_examples = 0
    labels_all: list[np.ndarray] = []
    probs_all: list[np.ndarray] = []

    for images, labels in loader:
        images = images.to(device)
        labels = labels.to(device)
        logits = model(images)
        loss = criterion(logits, labels)
        probs = torch.sigmoid(logits)
        total_loss += float(loss.item()) * images.size(0)
        total_examples += images.size(0)
        labels_all.append(labels.detach().cpu().numpy())
        probs_all.append(probs.detach().cpu().numpy())

    y_true = np.concatenate(labels_all, axis=0)
    y_prob = np.concatenate(probs_all, axis=0)
    return total_loss / max(total_examples, 1), y_true, y_prob


def save_curves(log: list[dict], output_path: Path) -> None:
    dataframe = pd.DataFrame(log)
    plt.figure(figsize=(9, 5))
    plt.plot(dataframe["epoch"], dataframe["train_loss"],
label="Train loss")
    plt.plot(dataframe["epoch"], dataframe["val_loss"],
label="Validation loss")
    if dataframe["val_macro_auroc"].notna().any():
        plt.plot(dataframe["epoch"],
dataframe["val_macro_auroc"], label="Validation macro AUROC")
    if dataframe["val_macro_f1"].notna().any():
        plt.plot(dataframe["epoch"], dataframe["val_macro_f1"],
label="Validation macro F1")
    plt.xlabel("Epoch")
    plt.grid(alpha=0.25)
    plt.legend()
    plt.tight_layout()
    plt.savefig(output_path, dpi=150)
    plt.close()


def main() -> None:
    args = parse_args()
    torch.manual_seed(args.seed)
    np.random.seed(args.seed)
    args.output_dir.mkdir(parents=True, exist_ok=True)

    device = torch.device("cuda" if torch.cuda.is_available()
else "cpu")
```

```
    train_dataset = maybe_limit(
        ChestXray14Dataset(args.csv, args.images_root,
split="train", transform=train_transforms()),
        args.limit,
    )
    val_dataset = maybe_limit(
        ChestXray14Dataset(args.csv, args.images_root,
split="val", transform=eval_transforms()),
        args.limit,
    )
    test_dataset = maybe_limit(
        ChestXray14Dataset(args.csv, args.images_root,
split="test", transform=eval_transforms()),
        args.limit,
    )

    if len(train_dataset) == 0 or len(val_dataset) == 0:
        raise RuntimeError("Train and validation splits must be
non-empty.")

    train_loader = DataLoader(
        train_dataset,
        batch_size=args.batch_size,
        shuffle=True,
        num_workers=args.num_workers,
        pin_memory=device.type == "cuda",
    )
    val_loader = DataLoader(
        val_dataset,
        batch_size=args.batch_size,
        shuffle=False,
        num_workers=args.num_workers,
        pin_memory=device.type == "cuda",
    )

    model = build_resnet50(num_classes=len(CLASSES),
pretrained=not args.no_pretrained).to(device)
    pretrained_imagenet_loaded = bool(getattr(model,
"loaded_imagenet_weights", False))
    if not pretrained_imagenet_loaded:
        print("WARNING: ResNet50 started without ImageNet
pretrained weights.")

    pos_weight = None if args.no_pos_weight else
compute_pos_weight(train_dataset, device)
    criterion = nn.BCEWithLogitsLoss(pos_weight=pos_weight)
    optimizer = torch.optim.AdamW(model.parameters(),
lr=args.lr)
    scheduler =
torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer,
mode="max", factor=0.5, patience=2)

    config = {
```

```
        "model": "resnet50",
        "classes": CLASSES,
        "dataset_csv": str(args.csv),
        "images_root": str(args.images_root),
        "epochs": args.epochs,
        "batch_size": args.batch_size,
        "lr": args.lr,
        "fallback_threshold": args.threshold,
        "seed": args.seed,
        "device": str(device),
        "train_size": len(train_dataset),
        "val_size": len(val_dataset),
        "test_size": len(test_dataset),
        "pretrained_imagenet_requested": not args.no_pretrained,
        "pretrained_imagenet_loaded":
pretrained_imagenet_loaded,
        "pos_weight_enabled": pos_weight is not None,
        "pos_weight": {
            class_name: float(pos_weight[index].detach().cpu())
if pos_weight is not None else 1.0
            for index, class_name in enumerate(CLASSES)
        },
    }
    (args.output_dir /
"config.json").write_text(json.dumps(config, ensure_ascii=False,
indent=2), encoding="utf-8")
    (args.output_dir /
"classes.json").write_text(json.dumps(CLASSES,
ensure_ascii=False, indent=2), encoding="utf-8")

    best_macro_auroc = -1.0
    best_metrics: dict = {}
    best_thresholds: dict[str, float] = {class_name:
args.threshold for class_name in CLASSES}
    log_rows: list[dict] = []

    for epoch in range(1, args.epochs + 1):
        train_loss = run_train_epoch(model, train_loader,
criterion, optimizer, device)
        val_loss, y_true, y_prob = evaluate_loader(model,
val_loader, criterion, device)
        thresholds = optimize_thresholds(y_true, y_prob)
        val_metrics = compute_multilabel_metrics(y_true, y_prob,
threshold=thresholds)
        val_macro_auroc = val_metrics.get("macro_auroc")
        scheduler.step(val_macro_auroc if val_macro_auroc is not
None else 0.0)

        log_row = {
            "epoch": epoch,
            "train_loss": train_loss,
            "val_loss": val_loss,
            "val_macro_auroc": val_macro_auroc,
```

```
            "val_macro_f1": val_metrics.get("macro_f1"),
            "lr": optimizer.param_groups[0]["lr"],
        }
        log_rows.append(log_row)
        print(
            f"Epoch {epoch}/{args.epochs} "
            f"train_loss={train_loss:.4f}
val_loss={val_loss:.4f} "
            f"val_macro_auroc={val_macro_auroc}
val_macro_f1={val_metrics.get('macro_f1')}"
        )

        comparable_auroc = val_macro_auroc if val_macro_auroc is
not None else -1.0
        if comparable_auroc > best_macro_auroc:
            best_macro_auroc = comparable_auroc
            best_metrics = val_metrics
            best_thresholds = thresholds
            torch.save(
                {
                    "model_state_dict": model.state_dict(),
                    "classes": CLASSES,
                    "config": config,
                    "metrics": val_metrics,
                    "thresholds": thresholds,
                },
                args.output_dir / "model_best.pt",
            )

    pd.DataFrame(log_rows).to_csv(args.output_dir /
"training_log.csv", index=False)
    (args.output_dir / "metrics_val.json").write_text(
        json.dumps(best_metrics, ensure_ascii=False, indent=2),
        encoding="utf-8",
    )
    (args.output_dir / "thresholds.json").write_text(
        json.dumps(best_thresholds, ensure_ascii=False,
indent=2),
        encoding="utf-8",
    )
    save_curves(log_rows, args.output_dir / "curves.png")
    print(f"Saved experiment to {args.output_dir}")
    print(f"Saved validation thresholds to {args.output_dir /
'thresholds.json'}")


if __name__ == "__main__":
    main()
```

Файл prepare_dataset.py

```
from __future__ import annotations
```

```
import argparse
import json
import re
from pathlib import Path

import numpy as np
import pandas as pd

from app.ml.classes import CLASSES, NO_FINDING_LABEL


def parse_args() -> argparse.Namespace:
    parser = argparse.ArgumentParser(description="Prepare
ChestX-ray14 metadata for multilabel training.")
    parser.add_argument("--metadata-csv", required=True,
type=Path)
    parser.add_argument("--images-root", required=True,
type=Path)
    parser.add_argument("--output-csv", required=True,
type=Path)
    parser.add_argument("--stats-json", type=Path)
    parser.add_argument("--seed", type=int, default=42)
    parser.add_argument("--train-ratio", type=float,
default=0.7)
    parser.add_argument("--val-ratio", type=float, default=0.15)
    parser.add_argument("--test-ratio", type=float,
default=0.15)
    parser.add_argument(
        "--keep-missing-images",
        action="store_true",
        help="Keep metadata rows even when the image file is
absent.",
    )
    return parser.parse_args()


def extract_patient_id(row: pd.Series) -> str | None:
    for column in ("Patient ID", "PatientID", "patient_id",
"Patient_ID"):
        if column in row and pd.notna(row[column]):
            return str(row[column])

    image_name = str(row.get("Image Index", row.get("image",
row.get("image_path", ""))))
    match = re.match(r"0*(\d+)_", Path(image_name).name)
    if match:
        return match.group(1)
    return None


def parse_labels(raw_labels: str) -> set[str]:
    labels = {label.strip() for label in
str(raw_labels).split("|") if label.strip()}
```

```
    if labels == {NO_FINDING_LABEL}:
        return set()
    return {label for label in labels if label in CLASSES}


def split_dataframe(
    dataframe: pd.DataFrame,
    seed: int,
    train_ratio: float,
    val_ratio: float,
    test_ratio: float,
) -> tuple[pd.DataFrame, str]:
    if not np.isclose(train_ratio + val_ratio + test_ratio,
1.0):
        raise ValueError("Train/val/test ratios must sum to
1.0")

    rng = np.random.default_rng(seed)
    dataframe = dataframe.copy()

    if dataframe["patient_id"].notna().all():
        keys = dataframe["patient_id"].astype(str).unique()
        split_strategy = "patient_id"
    else:
        keys = dataframe["image_path"].astype(str).unique()
        split_strategy = "image"
        print("WARNING: patient_id is unavailable for some rows;
falling back to image-level split.")

    shuffled = np.array(keys)
    rng.shuffle(shuffled)

    train_end = int(len(shuffled) * train_ratio)
    val_end = train_end + int(len(shuffled) * val_ratio)
    train_keys = set(shuffled[:train_end])
    val_keys = set(shuffled[train_end:val_end])

    key_column = "patient_id" if split_strategy == "patient_id"
else "image_path"

    def assign_split(value: str) -> str:
        value = str(value)
        if value in train_keys:
            return "train"
        if value in val_keys:
            return "val"
        return "test"

    dataframe["split"] =
dataframe[key_column].astype(str).map(assign_split)
    return dataframe, split_strategy
```

```
def prepare_metadata(
    metadata_csv: Path,
    images_root: Path,
    output_csv: Path,
    seed: int = 42,
    train_ratio: float = 0.7,
    val_ratio: float = 0.15,
    test_ratio: float = 0.15,
    filter_missing_images: bool = True,
) -> tuple[pd.DataFrame, dict]:
    source = pd.read_csv(metadata_csv)
    if "Finding Labels" not in source.columns:
        raise ValueError("ChestX-ray14 metadata must contain
'Finding Labels' column")

    image_column = "Image Index" if "Image Index" in
source.columns else "image_path"
    rows: list[dict] = []
    missing_images: list[str] = []
    for _, row in source.iterrows():
        image_name = Path(str(row[image_column])).name
        image_path = images_root / image_name
        image_exists = image_path.exists()
        if filter_missing_images and not image_exists:
            missing_images.append(image_name)
            continue

        labels = parse_labels(str(row["Finding Labels"]))
        label_values = {class_name: int(class_name in labels)
for class_name in CLASSES}

        try:
            output_image_path =
image_path.resolve().relative_to(output_csv.parent.resolve())
        except ValueError:
            output_image_path = Path(images_root.name) /
image_name

        rows.append(
            {
                "image_path":
str(output_image_path).replace("\\", "/"),
                "patient_id": extract_patient_id(row),
                **label_values,
            }
        )

    prepared = pd.DataFrame(rows)
    if prepared.empty:
        raise ValueError(
            "No rows left after filtering missing images. Check
--images-root or use --keep-missing-images."
        )
```

```
    prepared, split_strategy = split_dataframe(
        prepared,
        seed=seed,
        train_ratio=train_ratio,
        val_ratio=val_ratio,
        test_ratio=test_ratio,
    )

    stats = {
        "source_rows": int(len(source)),
        "total_images": int(len(prepared)),
        "available_image_rows": int(len(prepared)),
        "missing_image_rows": int(len(missing_images)),
        "missing_image_examples": missing_images[:20],
        "filtered_missing_images": bool(filter_missing_images),
        "classes": {class_name: int(prepared[class_name].sum())
for class_name in CLASSES},
        "split_counts":
prepared["split"].value_counts().to_dict(),
        "split_strategy": split_strategy,
        "no_finding_rows": int((prepared[CLASSES].sum(axis=1) ==
0).sum()),
        "warning": "No Finding is represented as a zero vector
across the 14 pathology classes.",
    }
    return prepared, stats


def main() -> None:
    args = parse_args()
    output_csv = args.output_csv
    output_csv.parent.mkdir(parents=True, exist_ok=True)
    stats_json = args.stats_json or output_csv.parent /
"chestxray14_stats.json"

    prepared, stats = prepare_metadata(
        metadata_csv=args.metadata_csv,
        images_root=args.images_root,
        output_csv=output_csv,
        seed=args.seed,
        train_ratio=args.train_ratio,
        val_ratio=args.val_ratio,
        test_ratio=args.test_ratio,
        filter_missing_images=not args.keep_missing_images,
    )
    prepared.to_csv(output_csv, index=False)
    stats_json.write_text(json.dumps(stats, ensure_ascii=False,
indent=2), encoding="utf-8")

    print(f"Saved prepared CSV: {output_csv}")
    print(f"Saved stats: {stats_json}")
    print(f"Source rows: {stats['source_rows']}")
```

```
    print(f"Available image rows:
{stats['available_image_rows']}")
    print(f"Filtered missing image rows:
{stats['missing_image_rows']}")
    for class_name, count in stats["classes"].items():
        print(f"{class_name}: {count}")
    print(f"Split counts: {stats['split_counts']}")
    print(f"Split strategy: {stats['split_strategy']}")


if __name__ == "__main__":
    main()
```

Файл homePage.tsx

```
import { useEffect, useState } from "react";
import { Activity, BarChart3 } from "lucide-react";

import {
  ClassificationResponse,
  ReportDetail,
  ReportSummary,
  classifyImage,
  fetchReport,
  fetchReports,
} from "../api/client";
import { HistorySidebar } from "../components/HistorySidebar";
import { ResultCard } from "../components/ResultCard";
import { UploadPanel } from "../components/UploadPanel";

function reportIdFromLocation(): string | null {
  const match =
window.location.pathname.match(/^\/reports\/([^/]+)$/);
  return match?.[1] ?? null;
}

export function HomePage() {
  const [file, setFile] = useState<File | null>(null);
  const [reports, setReports] = useState<ReportSummary[]>([]);
  const [result, setResult] = useState<ClassificationResponse |
ReportDetail | null>(null);
  const [activeReportId, setActiveReportId] = useState<string |
null>(() => reportIdFromLocation());
  const [isAnalyzing, setIsAnalyzing] = useState(false);
  const [error, setError] = useState<string | null>(null);

  useEffect(() => {
    loadReports();
  }, []);

  useEffect(() => {
    const handlePopState = () =>
setActiveReportId(reportIdFromLocation());
```

```
    window.addEventListener("popstate", handlePopState);
    return () => window.removeEventListener("popstate",
handlePopState);
  }, []);

  useEffect(() => {
    if (!activeReportId) {
      return;
    }
    fetchReport(activeReportId)
      .then((report) => {
        setResult(report);
        setError(null);
      })
      .catch((err: Error) => setError(err.message));
  }, [activeReportId]);

  async function loadReports() {
    try {
      setReports(await fetchReports());
    } catch (err) {
      setError(err instanceof Error ? err.message : "Не удалось
загрузить историю");
    }
  }

  async function handleAnalyze() {
    if (!file) {
      return;
    }
    setIsAnalyzing(true);
    setError(null);
    try {
      const response = await classifyImage(file);
      setResult(response);
      setActiveReportId(response.report_id);
      window.history.pushState({}, "",
`/reports/${response.report_id}`);
      await loadReports();
    } catch (err) {
      setError(err instanceof Error ? err.message : "Не удалось
выполнить анализ");
    } finally {
      setIsAnalyzing(false);
    }
  }

  function openReport(id: string) {
    setActiveReportId(id);
    window.history.pushState({}, "", `/reports/${id}`);
  }

  return (
```

```
    <main className="app-shell">
      <HistorySidebar reports={reports}
activeReportId={activeReportId} onOpenReport={openReport} />

      <section className="workspace">
        <header className="topbar">
          <div>
            <div className="brand-row">
              <Activity size={26} />
              <h1>MedVision CNN</h1>
            </div>
            <p>Классификация рентгеновских снимков грудной
клетки</p>
          </div>
          <div className="topbar-actions">
            <a className="button button--ghost"
href="/training">
              <BarChart3 size={18} />
              Метрики
            </a>
            <div className="research-notice">
              Исследовательский прототип. Результат не является
медицинским диагнозом.
            </div>
          </div>
        </header>

        <div className="analysis-layout">
          <UploadPanel
            file={file}
            isAnalyzing={isAnalyzing}
            error={error}
            onFileSelected={(selected) => {
              setFile(selected);
              setError(null);
            }}
            onAnalyze={handleAnalyze}
            onClear={() => setFile(null)}
          />
          <ResultCard result={result} />
        </div>
      </section>
    </main>
  );
}
```

ПРИЛОЖЕНИЕ Б

(обязательное)

Применение сверточных нейронных сетей для автоматической классификации медицинских снимков

Презентация

Применение сверточных нейронных сетей для автоматической классификации медицинских снимков

Автор работы:
студент группы 24ВВИм1 Шадрин Данила Алексеевич
Научный руководитель:
д.т.н., кафедра ВТ ПГУ Зинкин С.А.

Цель и задачи работы

Цель

разработать программный комплекс для автоматической классификации медицинских снимков на основе CNN

1. анализ рентгеновских снимков и датасета ChestX-ray14
2. выбор CNN-архитектуры и настройка ResNet50
3. предварительная обработка PNG, JPG и DICOM
4. Grad-CAM, отчет и web-интерфейс для проверки результата

Актуальность

! Медицинские изображения требуют внимательного анализа, а поток исследований растет. Часть рутинной проверки можно перенести на программные средства.

✓ рентгенограммы грудной клетки часто содержат несколько признаков одновременно

✓ CNN выделяет признаки автоматически, без ручного описания контуров и текстур

✓ Grad-CAM показывает области снимка, повлиявшие на вывод модели

Постановка задачи

Вход: рентгеновский снимок грудной клетки

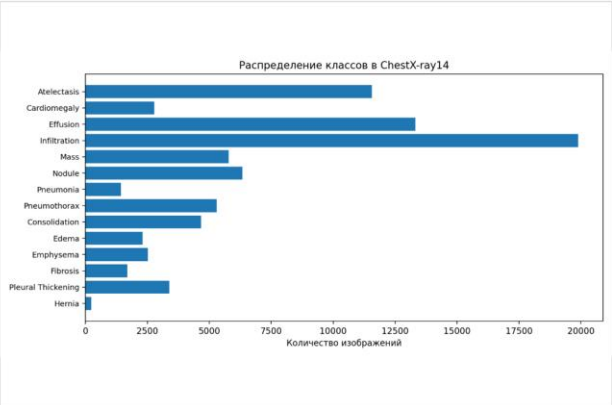
Выход: 14 независимых вероятностей + выбранные классы

Маршрут обработки изображения в исследовательском прототипе

Многометочная классификация

Один снимок может относиться сразу к нескольким классам, например Effusion и Cardiomegaly. Поэтому каждый класс оценивается отдельной сигмоидальной функцией и threshold.

Набор данных ChestX-ray14



14 классов

Atelectasis, Cardiomegaly, Effusion, Infiltration, Mass, Nodule, Pneumonia, Pneumothorax, Consolidation, Edema, Emphysema, Fibrosis, Pleural Thickening, Hernia

- разметка многометочная: один снимок может иметь несколько наблюдений
- классы распределены неравномерно, поэтому метрики считаются отдельно

Модель ResNet50

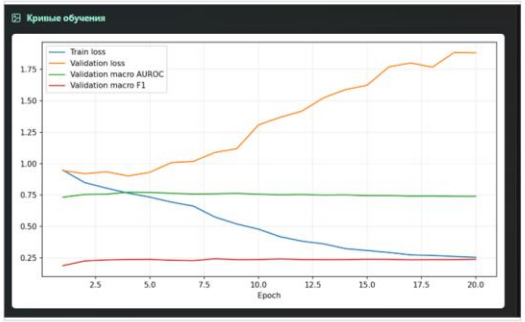
Почему ResNet50

- остаточные блоки помогают обучать глубокую сеть стабильнее
- предобученные веса позволяют применить перенос обучения
- последний слой заменен на 14 выходов ChestX-ray14

Preprocessing

- PNG, JPG и DICOM приводятся к единому виду
- изображение переводится в оттенки серого и 3 канала
- размер входа модели — 224 × 224

Обучение и оценка качества



Оценка не сводится к accuracy: из-за дисбаланса классов нужны AUROC, precision, recall / sensitivity, specificity и F1-score.

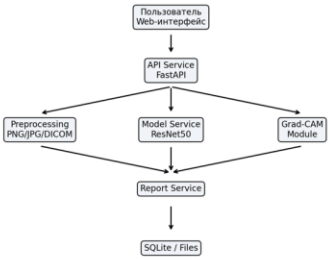
Метрики выводятся по каждому классу и вместе с thresholds — так проще понять, где модель работает уверенно, а где требуется настройка.

Архитектура приложения

Frontend взаимодействует с backend через REST API. Тяжелая логика — чтение снимков, инференс ResNet50 и Grad-CAM — выполняется на сервере.

React

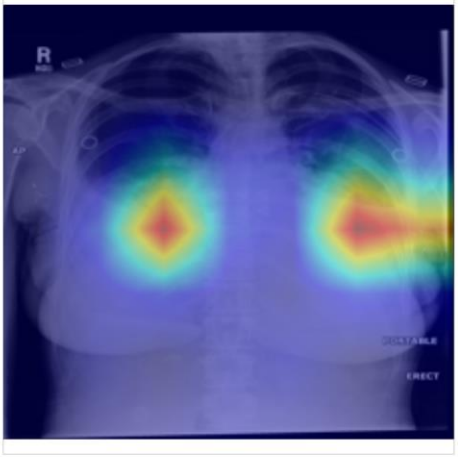
загрузка файла
отображение вероятностей
история отчетов



FastAPI + PyTorch

preprocessing
ResNet50
Grad-CAM
SQLite / Files

Результат анализа и Grad-CAM



Отчет сохраняет вероятности классов, версию модели, исходное изображение и тепловую карту Grad-CAM. Пользователь видит не новый инференс, а зафиксированный результат первичной обработки.

Проверка и ограничения

Проверено

- ✓ подготовка ChestX-ray14 и разбиение данных
- ✓ обучение ResNet50 и расчет метрик
- ✓ загрузка пользовательского снимка
- ✓ формирование Grad-CAM и сохранение отчета

Ограничения

- ChestX-ray14 содержит шумную разметку из текстовых отчетов
- редкие классы требуют отдельной настройки thresholds
- Grad-CAM объясняет поведение модели, но не заменяет медицинскую сегментацию
- перед практическим применением нужна независимая клиническая проверка

Заключение

- ✓ изучена задача автоматической классификации рентгеновских снимков грудной клетки
- ✓ разработан программный комплекс MedVision CNN с web-интерфейсом
- ✓ реализованы preprocessing, ResNet50, расчет метрик, Grad-CAM и хранение отчетов
- ✓ результат имеет исследовательский характер и не является медицинским диагнозом

Дальнейшая работа: улучшение качества по редким классам, ручной анализ Grad-CAM-карт, расширение тестового набора и проверка на независимых данных.