МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

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

МЕДИЦИНСКОЕ ПРИЛОЖЕНИЕ. МОДУЛЬ ПЕРВИЧНОЙ ДИАГНОСТИКИ ЗАБОЛЕВАНИЯ

Студент 11.06.24 Максимова Дарья Ренатовна
(подпись, дата) (ФИО полностью)

Руководитель 11.06.24 Митрохин М.А.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр 12.06.24 Бычков А.С.
(подпись, дата) (фамилия, инициалы)

Рецензент нач. отдела 13.06.24 Севастьянов А.Б.
(должность, место работы) (подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 27.05.24 № 11)

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

Работа защищена с отметкой отлично (протокол заседания ГЭК от 14.06.2024 № 22)

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

Пенза 2024

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

Факультет	*Кафедра*
вычислительной техники	«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
_____ М.А. Митрохин
«13» 09 2022 г.

ЗАДАНИЕ
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ

1. Студент Максимова Дарья Ренатовна гр. 22ВВИм1 факультета вычислительной техники

2.Тема работы «Медицинское приложение. Модуль первичной диагностики заболевания»

Тема утверждена приказом ПГУ №765/о от "13" сентября 2022 г.

3. Руководитель магистерской диссертации д.т.н., профессор Митрохин М.А.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Назначение разработки: определение заболеваний пользователей по изображениям. Модуль должен определять заболевание и искать локализацию на изображении. Разработать модуль первичной диагностики заболеваний по изображениям пользователей. В качестве пользовательского интерфейса использовать мобильное приложение. Модуль должен определять заболевания по изображениям пользователя, отправленным в чат-бот. После определения заболевания пользователя, система в интерфейсе мобильного чат- бота отправляет сообщение пользователю с названием заболевания и вероятностью определения этого заболевания

5. Перечень подлежащих разработке вопросов:
   1) Теоретический анализ предметной области использования нейронных сетей в сфере медицины
   2) Обзор и сравнительный анализ необходимых средств и инструментов для разработки модуля первичной диагностики
   3) Обучение нейронной сети для определения заболеваний по изображениям, отладка. Разработка и развертывание сервера. Создание модуля приложения.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области нейронных сетей в сфере медицины	10%	21.09.2022 – 21.11.2022	
2	Анализ предметной области мобильных приложений	5%	22.11.2022 – 22.01.2023	
3	Обучение нейронной сети для определения заболеваний по фото	35%	23.01.2023 – 23.04.2023	
4	Разработка модулей мобильного приложения	10%	24.04.2023 – 29.01.2024	
5	Разработка и отладка сервера	20%	30.01.2024 – 30.03.2024	
6	Оформление выпускной работы	20%	31.03.2024 – 31.05.2024	

Дата выдачи задания "13" 09 2022 г.

Руководитель магистерской диссертации _____ Митрохин М.А.
(подпись, дата)

Задание к исполнению принял студент _____ Максимова Д.Р.
(подпись, дата)

Работу к защите допустить
Декан факультета 13.06.2024 Фионова Л.Р.
(подпись, дата)

РЕФЕРАТ

Пояснительная записка содержит 76 листов, 30 рисунков, 3 таблицы, 29 источников, 2 приложения

ДЕТЕКЦИЯ ИЗОБРАЖЕНИЙ, R-CNN, PYTHON, DART, FLUTTER, НЕЙРОННЫЕ СЕТИ, МЕДИЦИНА

Цель работы – разработка программного средства распознавания заболеваний по изображениям. Данная разработка позволит пользователям своевременно диагностировать различные заболевания и получить рекомендации по лечению.

В результате проделанной работы были созданы мобильное приложение и нейронные сети. Мобильное приложение позволяет пользователю легко и быстро загружать изображения в нейронную сеть.

Для разработки использовался широкий список технологий: язык программирования Python, PyTorch Framework для разработки нейронной сети, фреймворк Flask для взаимодействие клиента и сервера, язык программирования Dart , фреймворк Flutter для создания мобильного приложения.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 6
1 Анализ предметной области 8
1.1 Общие характеристики нейронных сетей для обработки изображений 8
1.2 Применение нейронных сетей в медицине 10
1.3 Анализ моделей нейронных сетей 12
1.3.1 VGG 12
1.3.2 GoogLeNet 13
1.3.3 Resnet 16
1.4 Анализ темы 16
1.5 Анализ аналогов 21
1.6 Постановка задачи 24
1.7 Анализ клиент-серверной архитектуры в рамках решаемой задачи 26
1.8 Выбор языка программирования и среды разработки для клиентского приложения 29
1.9 Выбор фреймворка машинного обучения 31
1.10 Для разрабатываемого приложения самым подходящим фреймворком является PyTorch. 32
1.11 Выбор языка программирования и среды разработки для серверного приложения 32
1.12 Выводы 32
2 Разработка модуля первичной диагностики для мобильного приложения 33
2.1 Разработка серверной части для модуля первичной диагностики 33
2.1.1 Разработка серверного приложения 33
2.1.2 Развертывание серверного приложения 36
2.1.3 Контейнеризация приложения 37
ЗАКЛЮЧЕНИЕ 61
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 62

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей магистерской диссертации применяются следующие сокращения и обозначения

Детектор – техническое средство, которые указывает на наличие определенного свойства объекта

Датасет – набор данных

КТ – компьютерная томография

Рунет – русскоязычный интернет

Фреймворк – набор инструментов, ускоряющих разработку приложений

Чат-бот – программа, которая имитирует общение людей

Яндекс – поисковая система

API (Application Programming Interface) – программный интерфейс приложения

App Store – онлайн магазин приложений

CNN (Cellular neural network) – сверточная нейронная сеть

CPU (Central processing unit)– центральный процессор

Docker – платформа, позволяющая развертывать и управлять приложениями

Faster R-CNN Faster Regions with Convolutional Neural Network Features) – более быстрая сверточная нейронная сеть на основе регионов

Flask – фреймворк для создания приложений на языке Python

Git – распределенная система управления версиями

Python – язык программирования

R-CNN (Region cellular neural network) – алгоритм, основанный на сверточных нейронных сетях)

REST (Representational State Transfer) – архитектурный стиль, который определяет правила обмена данными между клиентом и сервером

RGB (red green blue) – цветовые каналы

RoI (Region Of Interest) – область интереса

RPN (Region Proposal Networks).

UML (Unified Modeling Language) – унифицированный язык моделирования

YAML (Yet Another Markup Language) – еще один язык разметки

ВВЕДЕНИЕ

В современном мире информационные технологии используются почти во всех сферах повседневной жизни.

Здоровье является самым важным компонентом для обеспечения качества жизни человека. Легкий доступ к информации в интернете, создает иллюзию того, что пользователи могут получить достоверную информацию. Но не вся информация в интернете является безопасной, особенно, если это касается здоровья человека. Человек, воспользовавшись рекомендациями из интернета по оказанию медицинской помощи, может навредить себе или своим близким. Интерес людей все больше растет к телемедицине. Люди хотят получать помощь онлайн. Но запрос получения медицинской помощи онлайн остается без удовлетворения, так как недостаточно интернет-площадок/интернет-сервисов для предоставления подобных услуг. Из-за слишком быстрых перемен в мире у многих людей появляются психологические травмы и прочие проблемы со здоровьем. Но некоторым людям бывает морально и физически тяжело дойти до поликлиники, для получения помощи от квалифицированного врача. Поэтому пользователи сети интернет прибегают к получению помощи онлайн. Ведь это быстро и человек в зависимости от своего ощущения в любой момент может отключить приложение, тем самым избежать состояние дискомфорта.

В магистерской диссертации объектом исследования является модуль медицинского виртуального помощника для диагностики заболеваний по изображениям, основанного на технологиях искусственного интеллекта. Медицинский помощник должен проводить первичную диагностику по фото, на основании изображений пользователя и давать рекомендации и информацию о первой помощи. Предметом исследования выступает модуль первичной диагностики заболеваний.

Цель магистерской диссертации является проектирование архитектуры, выявление и анализ требуемого функционала приложения, и реализация

модуля диагностики заболевания с помощью фотографий, основанного на технологиях искусственного интеллекта. Документирование разработанного модуля.

Задачи исследования:

– определение заболеваний по изображению пользователей,

– рекомендации, на основании анализа изображений пользователя.

1 Анализ предметной области

Прежде чем приступить к разработке программы необходимо изучить теорию по исследуемой теме. Для создания программы необходимо исследовать актуальность темы. Актуальность темы нужна, чтобы определить назначение программы, выявить проблему, которую будет решать разрабатываемая программа. Сделать сравнительный анализ аналогов, чтобы в своей работе учесть плюсы, а также избежать ошибок, которые были допущены в других программах. После этого необходимо определиться с технологиями, которые будут применяться в разработке. Для этого необходимо определить, какие продукты существуют для разработки, выявить плюсы и минусы различных инструментов и технологий. Сделать проектирование разрабатываемого сервиса, это поможет легко масштабировать сервис и избежать ошибок при эксплуатации.

После изучения и анализа предметной области, можно определиться с проблемой, которую решает продукт. Спроектировать функционал, который покрывает все потребности пользователя. На основе архитектуры и проекта приложения необходимо подобрать те инструменты и технологии, с помощью которых можно реализовать сервис быстро и с минимальным количеством ресурсов.

1.1 Общие характеристики нейронных сетей для обработки изображений

Для обработки изображений используются сверточные нейронные сети.

Сверточная нейронная сеть (CNN) – это класс алгоритмов машинного обучения, который хорошо подходит для анализа визуальных данных. Для алгоритмов CNN используются принципы линейной алгебры, в частности операции свертки, для извлечения признаков и идентификации шаблонов в изображениях.

Впервые термин «сверточный слой» ввел Ян Лекун[1] (ученый в области информатики) в конце 1990-х годов. В CNN сверточный слой соединяется только в локальной области с одним нейроном, как показано на рисунке 1.

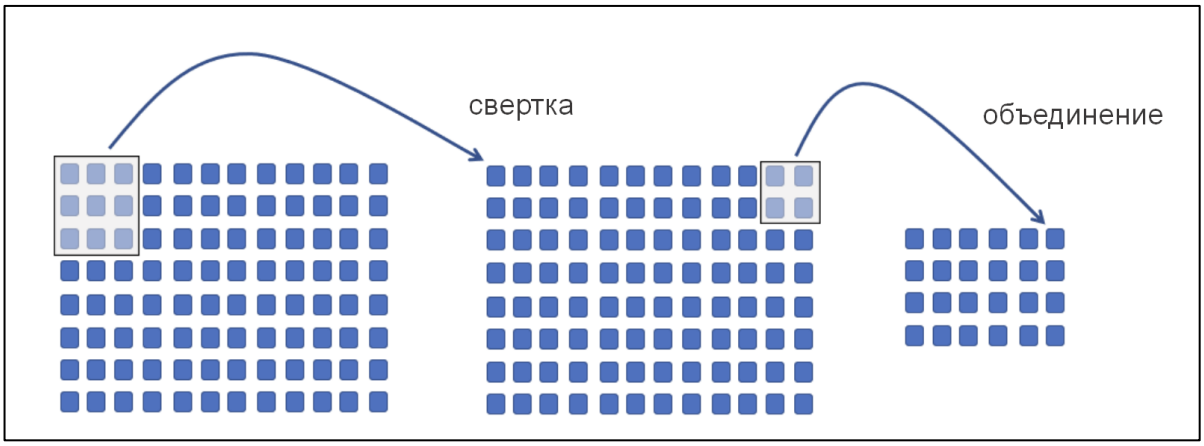


Рисунок 1 – Пример шага свертки

На первом уровне сверточной нейронной сети обрабатывается выходное изображение, разделенное на цветовые каналы (RGB). В зависимости от конфигурации, слой вычисляет от нескольких до многих так называемых карт объектов (центральная матрица на рисунке 1). В процессе обучения карты пространственных объектов учатся извлекать важные элементы из изображений. Например, они функционируют как определители границ областей для детекции или фильтры.

Современные CNN могу быть использованы для решения самых разнообразных задач.

Изначально поиск объектов на изображениях рассматривался как два процесса, не связанных друг с другом. Сначала в массиве прямоугольников, в которых, возможно, находится объект, определялись области. Затем составлялась карта объектов для областей и проводилась классификация.

R-CNN – один из первых двухступенчатых детекторов, который упрощает определение объектов на изображениях. Карта объектов вычисляется только один раз для всего изображения. Region of Interests (RoI) – этот слой определяет карты объектов стандартного размера для возможных

областей. Затем выполняется классификация и точное определение области, где находится объект. На рисунке 2 показана архитектура R-CNN.

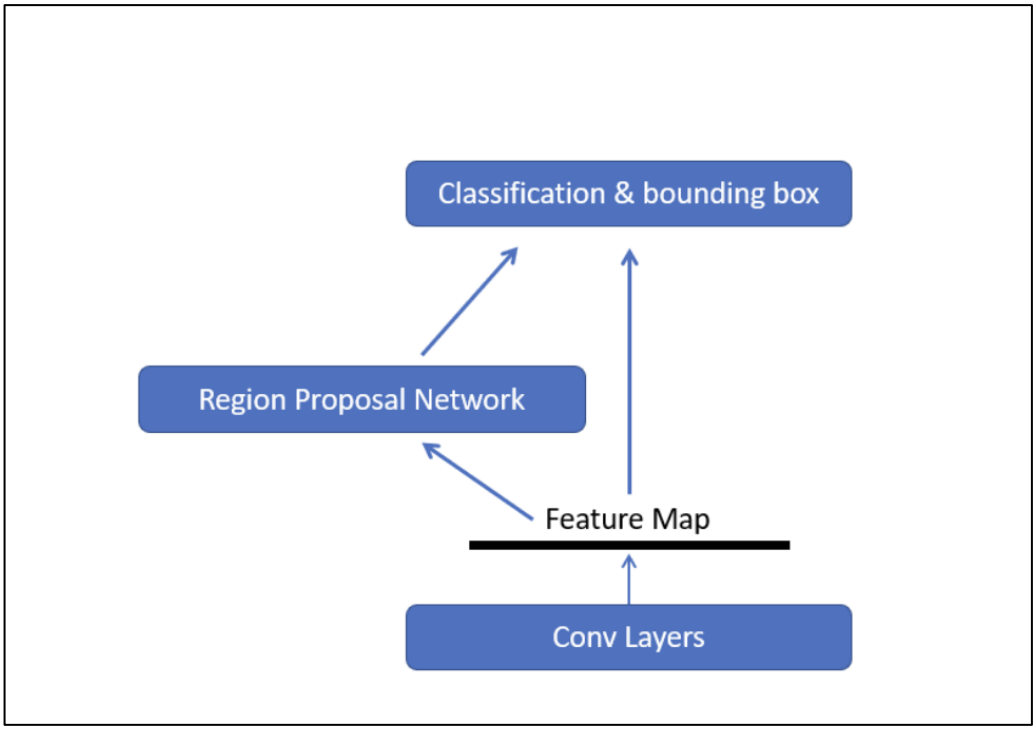


Рисунок 2 – Архитектура R-CNN

1.2 Применение нейронных сетей в медицине

В последние годы регионные сети (Region Proposal Networks, RPN) стали важной частью системы обнаружения объектов на основе глубокого обучения. Они позволяют эффективно и быстро генерировать предложения о регионах, что является важным шагом в обнаружении объектов

Для эффективной и точной генерации предложений о регионах была введена сеть Region Proposal Networks (RPN). Фаза генерации предложений регионов является в основном бесплатной из-за совместного использования свойств конволюционных свойств сети детекции, работающей вниз по потоку. Система обнаружения объектов на основе глубокого обучения теперь может функционировать на скорости кадров, почти реальном времени. Общая точность обнаружения объектов была увеличена за счет обученной RPN, которая также улучшает качество предложений регионов. R-CNN и fast R-CNN используют выборочную поиск, который медлен и требователен для

поиска предложений регионов, но faster R-CNN использует RPN, которая быстрее.

Для признания facial expressions была предложена faster R-CNN (Faster Regions with Convolutional Neural Network Features). Результаты экспериментов продемонстрировали эффективность и обобщаемость faster R-CNN для признания facial expressions. Число mAP составляло около 82%. Framework faster R-CNN был модифицирован для обнаружения объектов, зарытых под землей. Предложенная методика является довольно оптимистичной для работы с данными радара, проникающего в землю (GPR), даже с ограниченным количеством образцов обучения, согласно предварительным результатам обнаружения, которые демонстрируют значительные прогрессивные достижения по сравнению с классическими методами CV.

Результаты экспериментов продемонстрировали эффективность и обобщаемость faster R-CNN для признания facial expressions и обнаружения объектов, зарытых под землей. Модифицированная методика RPN может быть использована для различных приложений, требующих высокой точности и скорости обнаружения объектов.

Рак легких в его ранних стадиях проявляется в виде лёгких узлов, которые появляются в виде прозрачных или не прозрачных сферических тканевых образований в КТ-изображениях. Если узлы, появляющиеся в легких, могут быть диагностированы в ранней стадии и подозрительные пациенты могут быть отслежены и лечены, то распространение рака легких может быть эффективно уменьшено. В связи с широким распространением оборудования КТ, врачам необходимо наблюдать значительное количество изображений каждый день, и они могут ошибочно и пропустить поражения из-за продолжительного обзора результатов КТ.

Традиционные алгоритмы детекции узлов в легких делятся на две стадии: извлечение кандидатов узлов и исключение ложноположительных

узлов. Извлечение кандидатов узлов может быть основано на пороговом или алгоритме регионального роста. Однако, некоторые узлы имеют низкий контраст с окружающими тканями. Узлы, извлеченные этими методами, содержат значительное количество других маленьких регионов. Традиционные методы обнаружения неэффективны из-за продолжительных процессов, не позволяют выполнить детекцию в реальном времени и трудно обнаруживать более крупные данные. С продолжающимся улучшением производительности машинного оборудования и быстрым развитием глубокого обучения, различные методы обнаружения объектов широко используются в области обработки медицинских изображений. Поле обнаружения объектов обычно делят на модели обнаружения в одной стадии и модели обнаружения в двух стадиях. Faster R-CNN – модель обнаружения в двух стадиях основаны на сети региональных предложений для обнаружения. Введение сети региональных предложений в Faster R-CNN улучшает общую производительность и операционную скорость всего модели обнаружения и также имеет хороший результат в задаче обнаружения узлов.

1.3 Анализ моделей нейронных сетей

1.3.1 VGG

Модель VGG (Visual Geometry Group), разработанная Оксфордским университетом, является простотой и производительной [2]. Модель VGG создает простые сверточные слои размером 3x3. Такой выбор архитектуры позволил сети изучить сложные функции при сохранении эффективности вычислений, продемонстрировав, что высокая производительность может быть достигнута без необходимости усложнения архитектуры. Преимуществом VGG16 является простота, что означает, что его легко понять и реализовать, в то время как недостатком является то, что модель имеет относительно большее количество параметров, что приводит к большим затратам на хранение и вычисление данных. Архитектура VGG представлена на рисунке 3.

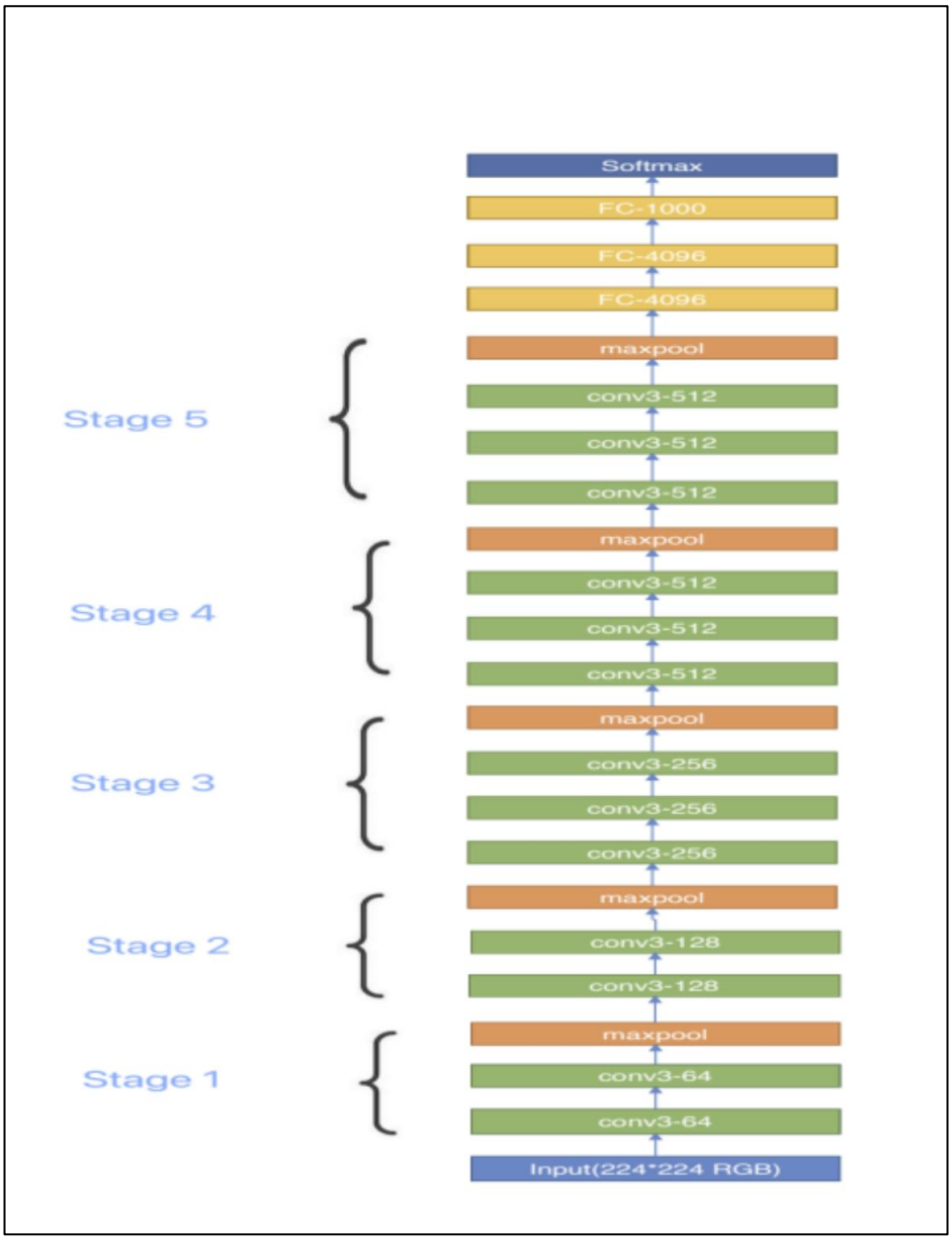


Рисунок 3 – Архитектура VGG

1.3.2 GoogLeNet

GoogleNet — это глубокая конволюционная нейронная сеть, которая была разработана компанией Google для решения задачи классификации изображений. Архитектура GoogleNet основана на глубокой структуре, которая включает в себя несколько уровней конволюционных слоев, последовательно соединенных с полносочетными слоями. Это позволяет нейронной сети эффективно извлекать фичи из изображений и улучшать ее способность к классификации.

Начальный слой архитектуры GoogLeNet, называемый «базовым слоем», включает в себя обычный сверточный слой, на смену которому приходят операции объединения. Эта структура служит для сжатия пространственных размеров входного изображения при одновременном увеличении глубины. В основе архитектуры GoogLeNet лежит модуль Inception, который использует распараллеленный сверточные фильтры различной размерности, а также операции максимального объединения в пул. Такая конструкция позволяет сети фиксировать иерархии пространственных объектов в различных масштабах. В отличие от предыдущих архитектур, таких как AlexNet и VGG, которые включали полностью подключенные уровни, GoogLeNet использует средний уровень объединения, предшествующий конечному уровню softmax. Этот выбор приводит к существенному сокращению общего количества параметров. На этапе обучения архитектура дополнительно дополняется двумя вспомогательными классификаторами, добавляемыми к промежуточным уровням. Эти классификаторы предоставляют дополнительные градиенты, тем самым повышая эффективность обратного распространения. Вспомогательные классификаторы опущены на этапе логического вывода для удобства вычислений. Архитектура GoogLeNet представлена на рисунке 4.

GoogLeNet может иметь низкую точность на определенных типах задач, что может ограничивать ее использование в медицинской сфере.

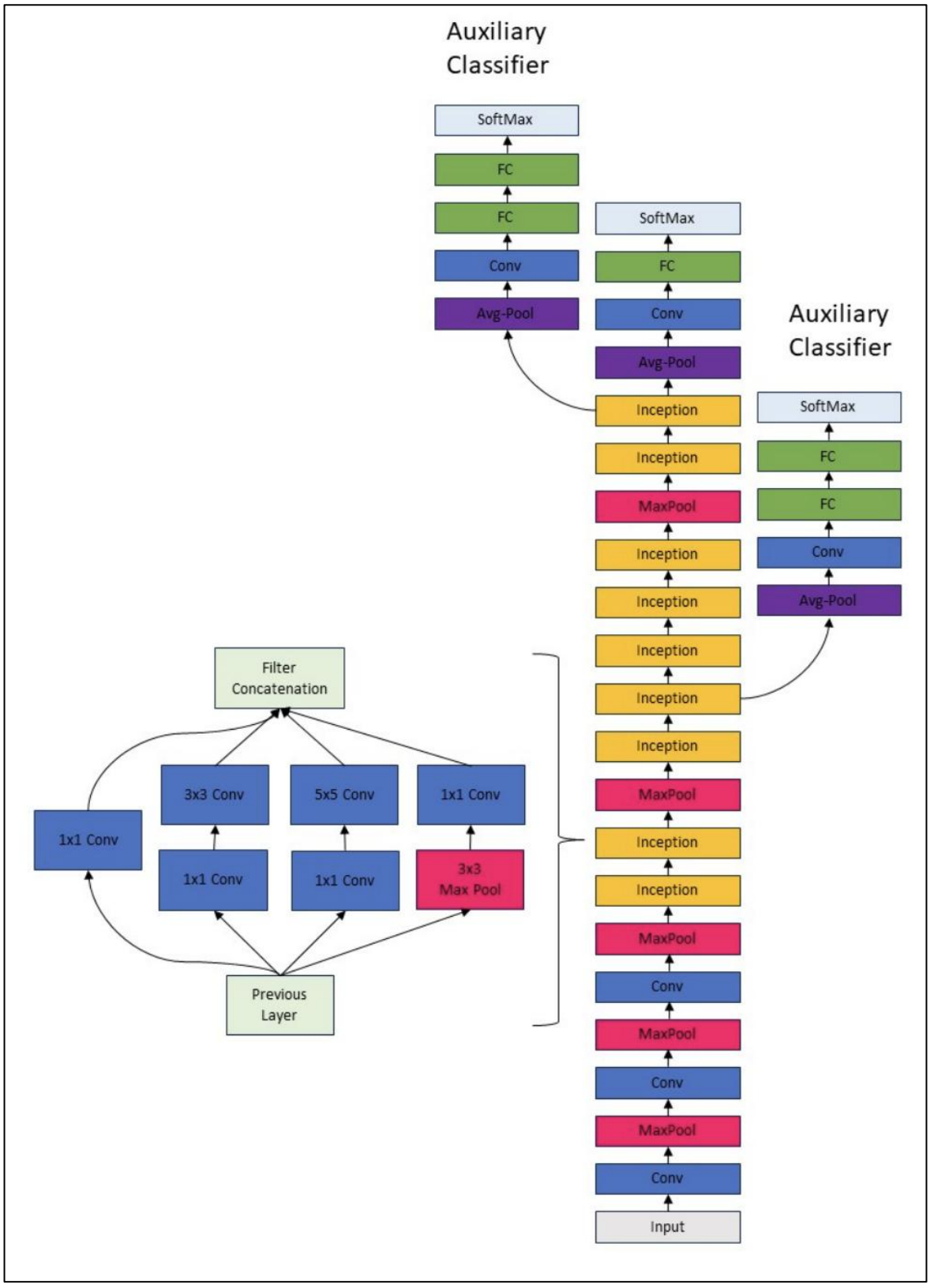


Рисунок 4 – Архитектура GoogLeNet

1.3.3 Resnet

Сеть ResNet основана на сети VGG19, с введением остаточные единицы через механизм быстрого соединения [3]. Это решает проблему деградации, связанную с уменьшением уровня точности при увеличении уровней сети, путем сопоставления идентификаторов (что относится к изогнутым линиям) и остаточного отображения (оставшейся части модели). В модели ResNet очень важным принципом проектирования является то, что количество карт объектов удваивается, когда размер карты объектов уменьшается вдвое. Этот принцип помогает поддерживать сложность сети. ResNet создает остаточное обучение, добавляя механизм короткого соединения между каждыми двумя слоями. При такой стратегии применение заполнения нулями увеличивает размеры и не более того. аргументы появятся во время сети.

Архитектура представлена на рисунке 5 ниже.

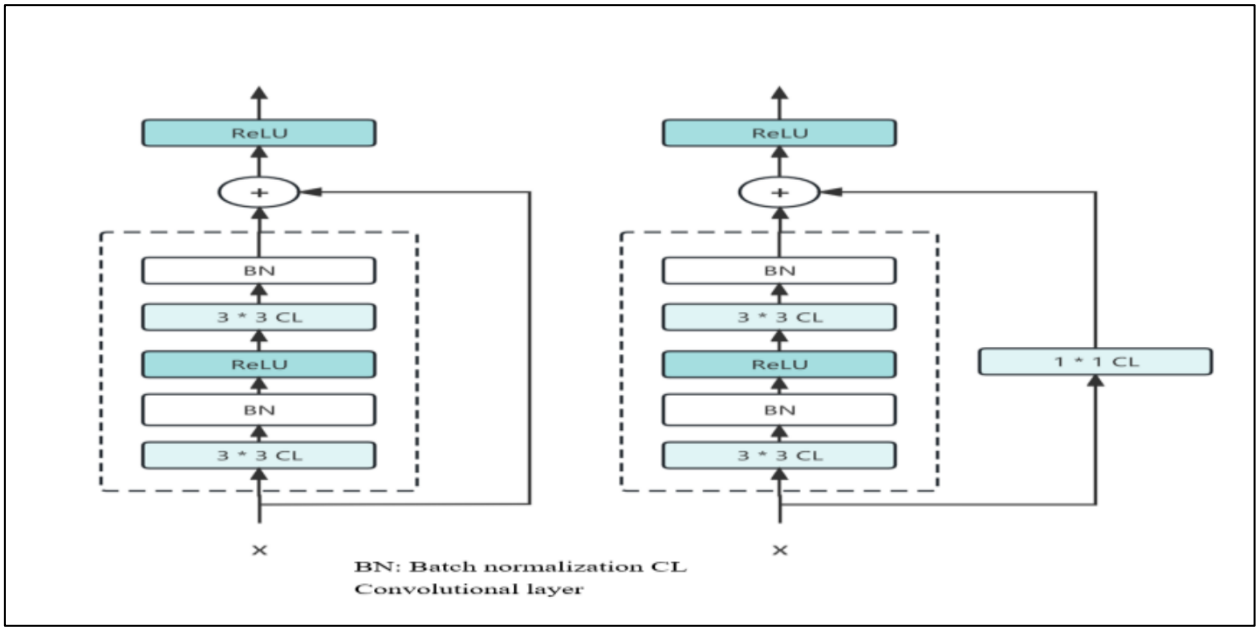


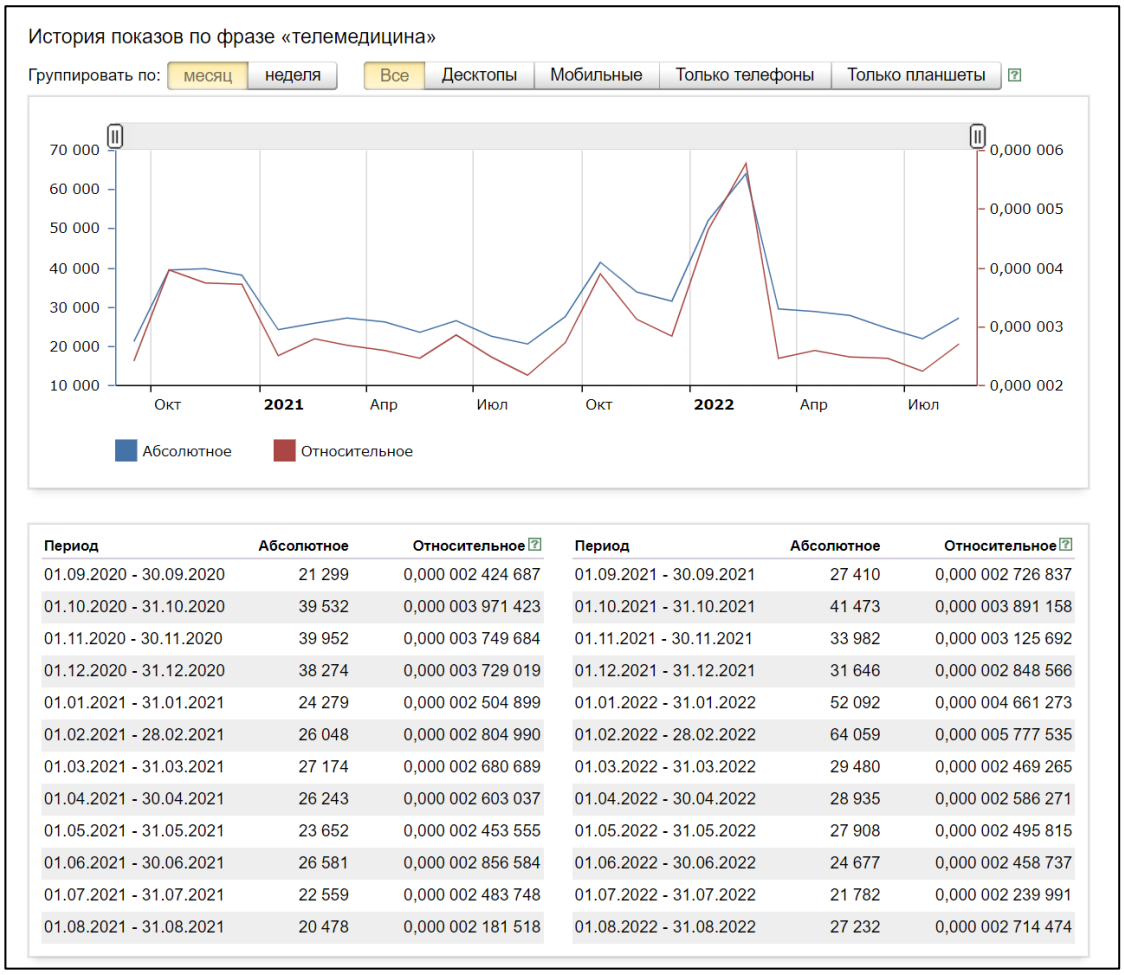
Рисунок 5 – Архитектура сети

1.4 Анализ темы

Люди все больше используют информационные технологии для решения своих повседневных проблем. Легкий доступ к информации в интернете, создает иллюзию того, что пользователи могут получить

квалифицированную информацию в интернете. Но не вся информация в интернете является безопасной, особенно, если это касается здоровья человека. Человек, воспользовавшись рекомендациями из интернета по оказанию медицинской помощи, может навредить себе или своим близким. Мною было проанализировано множество интернет-ресурсов с рекомендациями по оказанию медицинской помощи, была выявлена проблема: не существует сервисов, которые осуществляли бы подбор медицинских рекомендаций для каждого пользователя индивидуально.

Интерес людей все больше растет к телемедицине. Люди хотят получать помощь онлайн. Но запрос получения медицинской помощи онлайн остается без удовлетворения, так как недостаточно интернет-площадок/интернет-сервисов для предоставления подобных услуг. История показов в поисковой системе «Яндекс» по фразе «телемедицина» показана на рисунке 6.

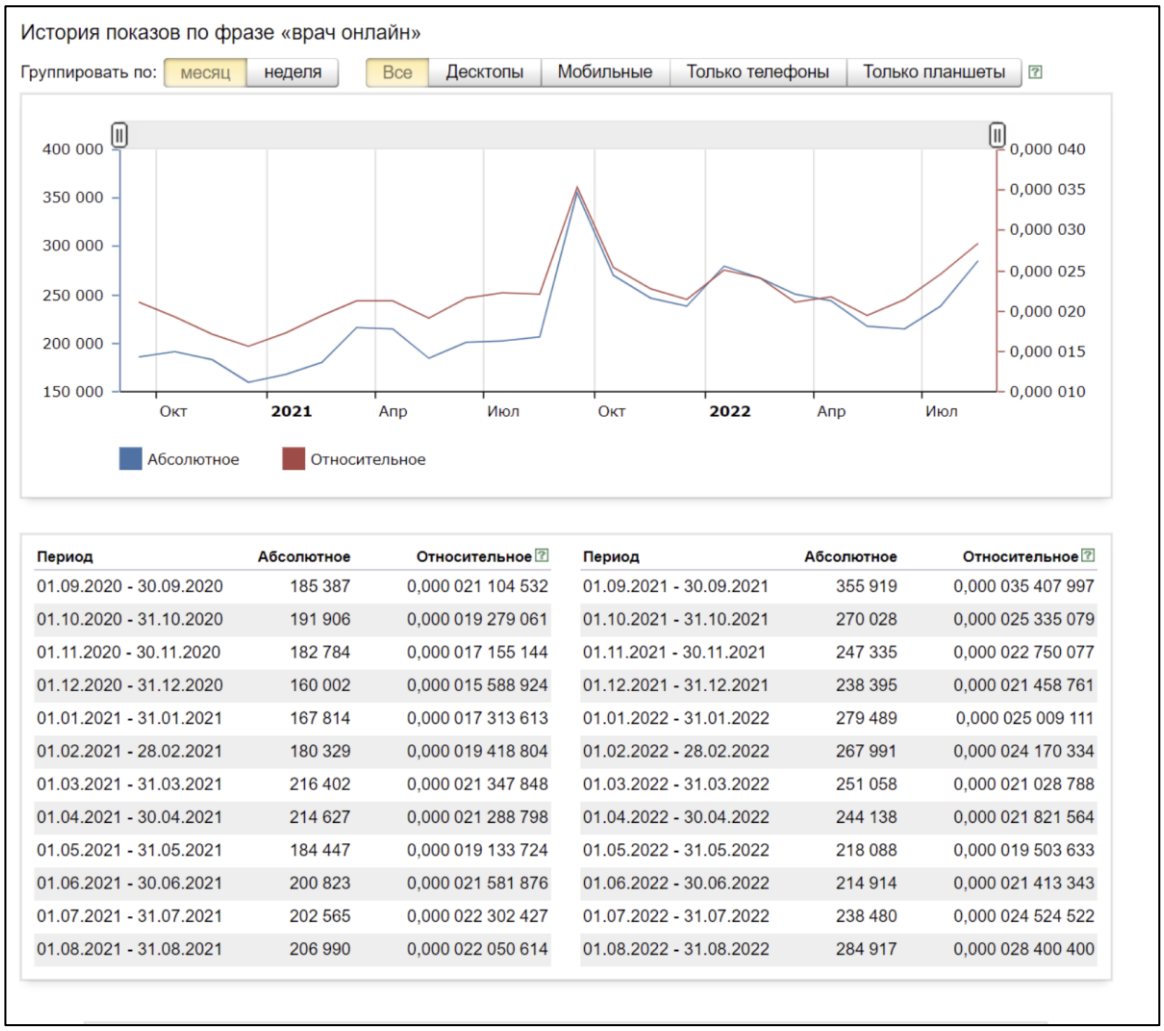


Период	Абсолютное	Относительное	Период	Абсолютное	Относительное
01.09.2020 - 30.09.2020	21 299	0,000 002 424 687	01.09.2021 - 30.09.2021	27 410	0,000 002 726 837
01.10.2020 - 31.10.2020	39 532	0,000 003 971 423	01.10.2021 - 31.10.2021	41 473	0,000 003 891 158
01.11.2020 - 30.11.2020	39 952	0,000 003 749 684	01.11.2021 - 30.11.2021	33 982	0,000 003 125 692
01.12.2020 - 31.12.2020	38 274	0,000 003 729 019	01.12.2021 - 31.12.2021	31 646	0,000 002 848 566
01.01.2021 - 31.01.2021	24 279	0,000 002 504 899	01.01.2022 - 31.01.2022	52 092	0,000 004 661 273
01.02.2021 - 28.02.2021	26 048	0,000 002 804 990	01.02.2022 - 28.02.2022	64 059	0,000 005 777 535
01.03.2021 - 31.03.2021	27 174	0,000 002 680 689	01.03.2022 - 31.03.2022	29 480	0,000 002 469 265
01.04.2021 - 30.04.2021	26 243	0,000 002 603 037	01.04.2022 - 30.04.2022	28 935	0,000 002 586 271
01.05.2021 - 31.05.2021	23 652	0,000 002 453 555	01.05.2022 - 31.05.2022	27 908	0,000 002 495 815
01.06.2021 - 30.06.2021	26 581	0,000 002 856 584	01.06.2022 - 30.06.2022	24 677	0,000 002 458 737
01.07.2021 - 31.07.2021	22 559	0,000 002 483 748	01.07.2022 - 31.07.2022	21 782	0,000 002 239 991
01.08.2021 - 31.08.2021	20 478	0,000 002 181 518	01.08.2022 - 31.08.2022	27 232	0,000 002 714 474

Рисунок 6 – История показов по фразе «Телемедицина»

На графике видно, что интерес людей сильно растет к телемедицине, особенно в зимний период. Как правило, это связано с погодными условиями (гололедица, мороз, сезонные заболевания). Например, количество поисков в феврале 2022 года почти в 2,5 раза больше, чем количество поисков в феврале 2021. Это свидетельствует о том, что все больше людей узнают и интересуются о возможности получения помощи онлайн.

История показов в поисковой системе «Яндекс» по фразе «врач онлайн» показана на рисунке 7.



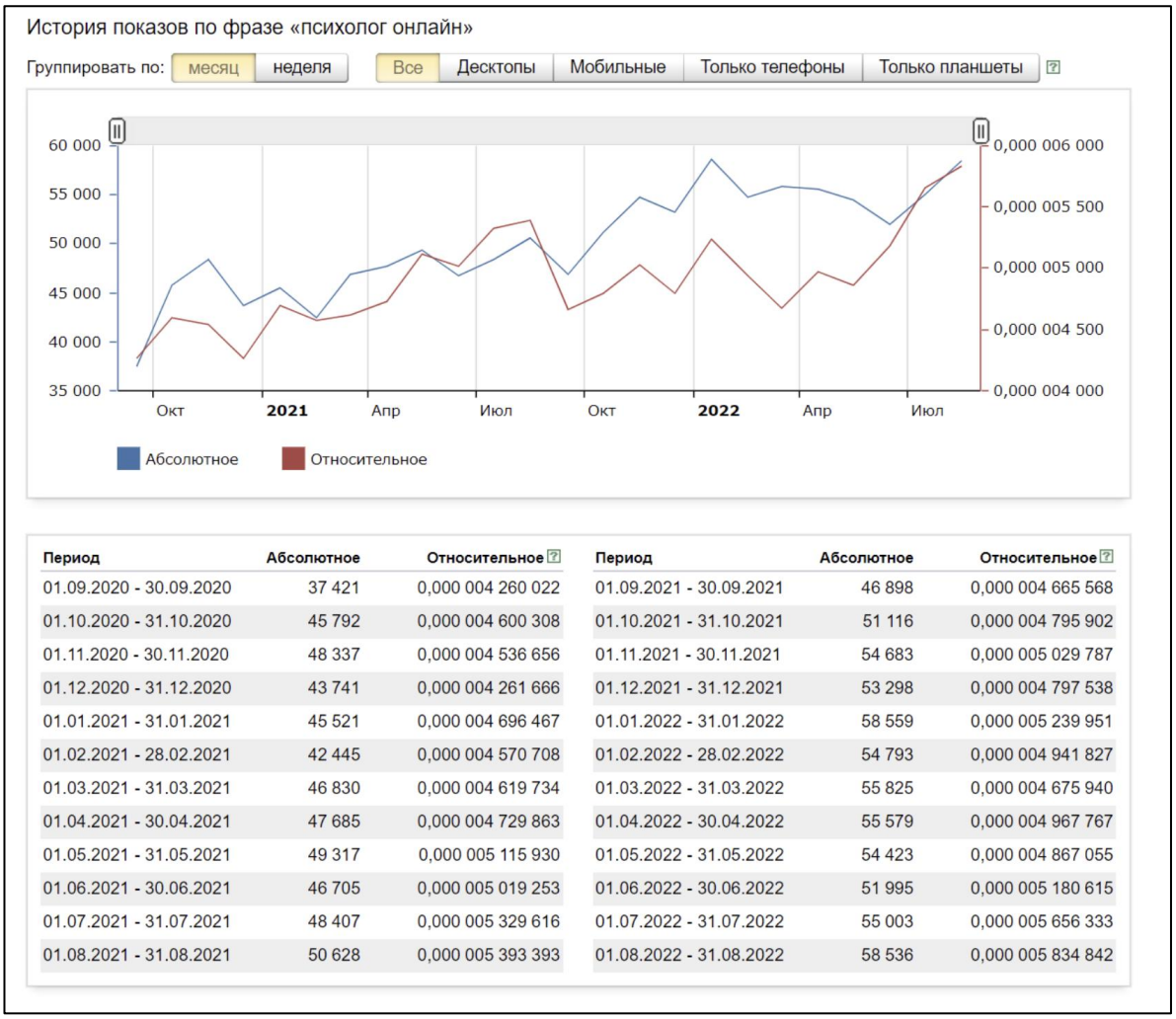
Период	Абсолютное	Относительное
01.09.2020 - 30.09.2020	185 387	0,000 021 104 532
01.10.2020 - 31.10.2020	191 906	0,000 019 279 061
01.11.2020 - 30.11.2020	182 784	0,000 017 155 144
01.12.2020 - 31.12.2020	160 002	0,000 015 588 924
01.01.2021 - 31.01.2021	167 814	0,000 017 313 613
01.02.2021 - 28.02.2021	180 329	0,000 019 418 804
01.03.2021 - 31.03.2021	216 402	0,000 021 347 848
01.04.2021 - 30.04.2021	214 627	0,000 021 288 798
01.05.2021 - 31.05.2021	184 447	0,000 019 133 724
01.06.2021 - 30.06.2021	200 823	0,000 021 581 876
01.07.2021 - 31.07.2021	202 565	0,000 022 302 427
01.08.2021 - 31.08.2021	206 990	0,000 022 050 614

Период	Абсолютное	Относительное
01.09.2021 - 30.09.2021	355 919	0,000 035 407 997
01.10.2021 - 31.10.2021	270 028	0,000 025 335 079
01.11.2021 - 30.11.2021	247 335	0,000 022 750 077
01.12.2021 - 31.12.2021	238 395	0,000 021 458 761
01.01.2022 - 31.01.2022	279 489	0,000 025 009 111
01.02.2022 - 28.02.2022	267 991	0,000 024 170 334
01.03.2022 - 31.03.2022	251 058	0,000 021 028 788
01.04.2022 - 30.04.2022	244 138	0,000 021 821 564
01.05.2022 - 31.05.2022	218 088	0,000 019 503 633
01.06.2022 - 30.06.2022	214 914	0,000 021 413 343
01.07.2022 - 31.07.2022	238 480	0,000 024 524 522
01.08.2022 - 31.08.2022	284 917	0,000 028 400 400

Рисунок 7 - История показов по фразе «Врач онлайн»

Исходя из графика видно, что многие люди ищут медицинскую помощь через интернет. Можно сделать вывод, что для многих людей получение медицинской помощи онлайн актуальная проблема. Интерес пользователей к получению медицинской помощи онлайн вырос почти в 2 раза (сентябрь 2021/сентябрь 2022)

История показов в поисковой системе «Яндекс» по фразе «психолог онлайн» показана на рисунке 8.



Период	Абсолютное	Относительное	Период	Абсолютное	Относительное
01.09.2020 - 30.09.2020	37 421	0,000 004 260 022	01.09.2021 - 30.09.2021	46 898	0,000 004 665 568
01.10.2020 - 31.10.2020	45 792	0,000 004 600 308	01.10.2021 - 31.10.2021	51 116	0,000 004 795 902
01.11.2020 - 30.11.2020	48 337	0,000 004 536 656	01.11.2021 - 30.11.2021	54 683	0,000 005 029 787
01.12.2020 - 31.12.2020	43 741	0,000 004 261 666	01.12.2021 - 31.12.2021	53 298	0,000 004 797 538
01.01.2021 - 31.01.2021	45 521	0,000 004 696 467	01.01.2022 - 31.01.2022	58 559	0,000 005 239 951
01.02.2021 - 28.02.2021	42 445	0,000 004 570 708	01.02.2022 - 28.02.2022	54 793	0,000 004 941 827
01.03.2021 - 31.03.2021	46 830	0,000 004 619 734	01.03.2022 - 31.03.2022	55 825	0,000 004 675 940
01.04.2021 - 30.04.2021	47 685	0,000 004 729 863	01.04.2022 - 30.04.2022	55 579	0,000 004 967 767
01.05.2021 - 31.05.2021	49 317	0,000 005 115 930	01.05.2022 - 31.05.2022	54 423	0,000 004 867 055
01.06.2021 - 30.06.2021	46 705	0,000 005 019 253	01.06.2022 - 30.06.2022	51 995	0,000 005 180 615
01.07.2021 - 31.07.2021	48 407	0,000 005 329 616	01.07.2022 - 31.07.2022	55 003	0,000 005 656 333
01.08.2021 - 31.08.2021	50 628	0,000 005 393 393	01.08.2022 - 31.08.2022	58 536	0,000 005 834 842

Рисунок 8 - История показов по фразе «Психолог онлайн»

Из-за слишком быстрых перемен окружающей обстановки в мире у многих людей появляются психологические проблемы. Но некоторым людям бывает морально и физически тяжело дойти для поликлиники, для получения психологической помощи от квалифицированного врача. Поэтому пользователи сети интернет прибегают к получению психологической помощи онлайн. Ведь человек в зависимости от своего ощущения в любой момент может отключить аудио/видео связь, тем самым избежать состояние дискомфорта. На графике видно, что интерес людей к получению психологической помощи онлайн постоянно растет.

Основываясь на вышеприведенных данных, было принято решение разработать такую систему, которая удовлетворяет запрос пользователей.

1.5 Анализ аналогов

Множество крупных российских компаний уже начали разработку ресурсов, осуществляющих первичную диагностику пользователя.

AI Skin (согласно описанию разработчика приложения на официальном сайте) помогает человеку определить заболевания кожи по фото с помощью ИИ. Программа по фотографии распознает только 7 типов кожных высыпаний: меланомы, базалиомы, меланоцитарные невусы, актинические и себорейные кератозы, сосудистые высыпания и дерматофибромы. В описании программы указано, что пациент или врач должны загрузить фото через онлайн-сервис или мобильное приложение. После чего алгоритм ИИ анализирует снимки и сопоставляет с базой данных для диагностики заболевания. В результате программа выдает предполагаемый диагноз. Никаких других данных от пользователя не требуется. Доступ мобильного приложения есть для ОС Android и iOS. Мобильное приложение полностью бесплатное. В программе доступен только 1 язык – русский. Самым главным минусом этого сервиса является его неработоспособность. При загрузке фотографии в приложение, система отображает информационное окно об ошибке. По этой причине, в магазине приложений AppStore не пользуется популярностью среди пользователей и имеет низкий рейтинг. Интерфейс приложения представлен на рисунке 9.

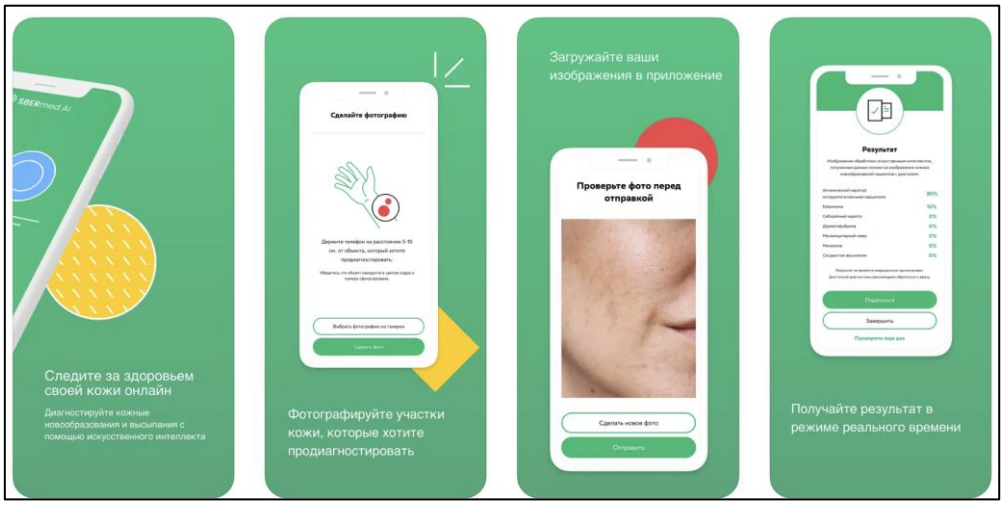


Рисунок 9 – Интерфейс приложения AI Skin

CheckDerm – проект, специализирующийся на диагностике кожных заболеваний по фотографии. Разработчик утверждает, что программа диагностирует 8 видов заболеваний: ветряная оспа, крапивница, атопический дерматит, монетовидная экзема, корь, краснуха, псориаз, укусы насекомых. Для получения результата диагностики пользователю необходимо пройти тест, уточняющий вопросы по проблеме пациента. После чего пользователь загружает фотографию пораженного участка кожи. В данный момент для диагностики заболеваний доступная только web-версия сервиса. После получения результатов диагностики, пользователь получает актуальную информацию о своем предполагаемом заболевании (на основании результатов теста и диагностики по фото), а также записаться на очный прием к врачу, либо на онлайн-консультацию с врачом. На сайте присутствует много рекламы, которая создает визуальный шум. Плохо, что у сервиса нет мобильного приложения, т.к. доля мобильного трафика в Рунете составляет 83% (на 2021 год). Интерфейс web-сайта представлен на рисунке 10.

Рисунок 10 - Интерфейс сайта CheckDerm

VisualDx – это визуальный справочный инструмент для медицинских работников, который помогает принимать терапевтические решения. Приложение доступно на мобильных устройствах – Android, iOS. Имеет бесплатный пробный период. Приложение не ориентированно на людей без медицинского образования. Приложение хорошо подходит для студентов медицинских ВУЗов, т.к. в приложении пользователь имеет доступ к обширному количеству профессиональной литературы. В приложении осуществляется диагностика более чем по 3200 диагнозам. Интерфейс представлен ниже на рисунке 11.

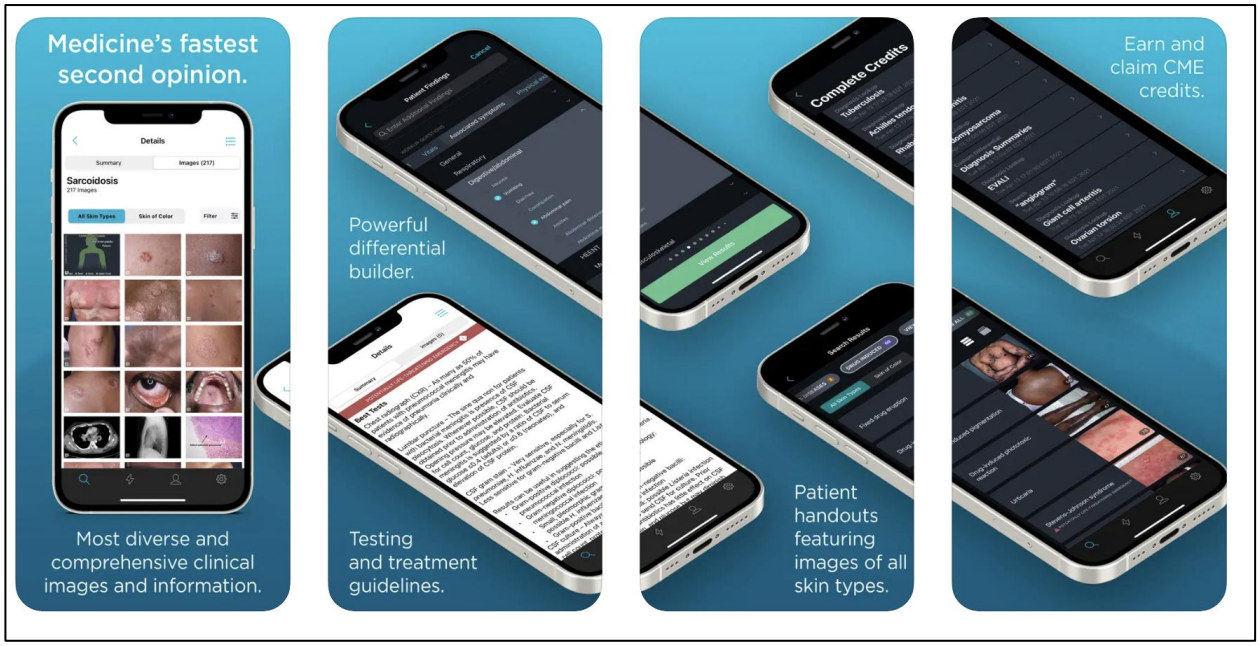


Рисунок 11 - Интерфейс приложения VisualDx

Результат сравнения функционала рассмотренных сервисов представлен в таблице 1.

В ходе анализа было выявлено, что присутствует дефицит сервисов, осуществляющих диагностику заболеваний по фото. VisualDx обладает большой базой заболеваний, но этот сервис ориентирован на сотрудников здравоохранений, не подходит для пользователей, не обладающих знаниями в сфере медицины.

В целом, дефицит сервисов, которые могут помочь пользователям диагностировать заболевания по фото, является значительной проблемой в

области компьютерного зрения и медицины. Необходимо разработать новые сервисы, которые будут доступны для широкой аудитории и не требовать специальных знаний в области медицины.

Таблица 1 – Результат сравнительного анализа сервисов

Функция	Название приложения		
	AI Skin	*CheckDerm*	*VisualDx*
Бесплатное использование	+	+	-
Первичная диагностика заболеваний	-	+	+
Работоспособность сервиса	-	-	-
Отслеживание самочувствия	-	-	-
Интуитивный интерфейс	-	-	-
Наличие мобильного приложения	+	-	+
Возможность записаться к доктору онлайн	-	+	-
Информация о предполагаемом заболевании	-	+	+
Количество заболеваний	7	8	3200

На таблице видно, что ни одно из приложений не удовлетворит запрос пользователей. Для пользователей необходимо реализовать приложение с интуитивно понятным интерфейсом, быстрым и точным анализом по изображениям пользователя

1.6 Постановка задачи

Требуется разработать модули в мобильном приложении, а также подсистему для анализа изображения пользователя для выявления и диагностирования различных заболеваний. В описанную подсистему должна входить сама нейронная сеть, требующаяся для анализа, а также сервер для обеспечения взаимодействия с общей системой.

Клиентское мобильное приложение должно обладать следующим функционалом:

– напоминание о приеме лекарств,

– звонок с виртуальным помощником, взаимодействие с которым будет осуществляться посредством общения голосом. Ответы от такого помощника также должны воспроизводиться в звуковом формате.

Подсистема будет использована для взаимодействия с общей системой, которая необходима для комплексного анализа и диагностики пользователя. Система будет включать в себя чат-ботов и работать в режиме реального времени. Будет состоять из трех нейронных сетей:

– анализ изображения. Нейронная сеть должна выполнять обработки фотографии от пользователя и искать место локализации болезни. Результатом обработки является набор координат ограничивающей рамки, а также процент, показывающий вероятность наличия болезни,

– анализ настроения. Данная нейронная сеть выполняет анализ текста, выполняя классификацию по типу настроения,

– языковая модель для генерации текста. Данная модель проводит анализ текстового сообщения от пользователя, выдавая ему рекомендации по решению его проблемы.

Выделение сервера для анализа изображений и настройка доступа к ней с помощью REST API обеспечат улучшение результата выдачи генератора текста. Использование алгоритмов машинного зрения и глубокого обучения

для анализа изображений и результатов анализа для генерации текста позволит создать более точный и информативный текст.

Все вышеописанные модели должны быть взаимосвязаны друг с другом для улучшения результата выдачи генератора текста. Поэтому необходимо вывести подсистему для анализа изображений на выделенный сервер и настроить доступ с помощью REST API.

Также необходимо обеспечить максимальное быстродействие и отзывчивость подсистемы.

1.7 Анализ клиент-серверной архитектуры в рамках решаемой задачи

Существует большое количество подходов для построение сложных систем, все они задают способы разбиения приложения на модули. Хорошая архитектура должна обладать следующими свойствами:

– возможность тестирования системы,

– независимость архитектуры от любого окружения (база данных, интерфейс и другие службы).

Клиент-серверная архитектура удовлетворяет этим требованиям.

Для построения клиент-серверной архитектуры необходимо разделить систему на две категории программ: клиентская программу и сервер.

Клиентская программа подключается к серверу и выполняет поставленную задачу. В частности, это предоставление некоторых данных с сервера. Клиент будет взаимодействовать с сервером для того, чтобы производить анализ изображений с помощью нейронной сети. Взаимодействие осуществляется через интернет.

Для обработки запросов по диагностике заболеваний с помощью изображений необходимо осуществить взаимодействие клиента и сервера. Взаимодействие будет проходить через сеть «Интернет». UML-диаграмма последовательности, представленная на рисунке 12.

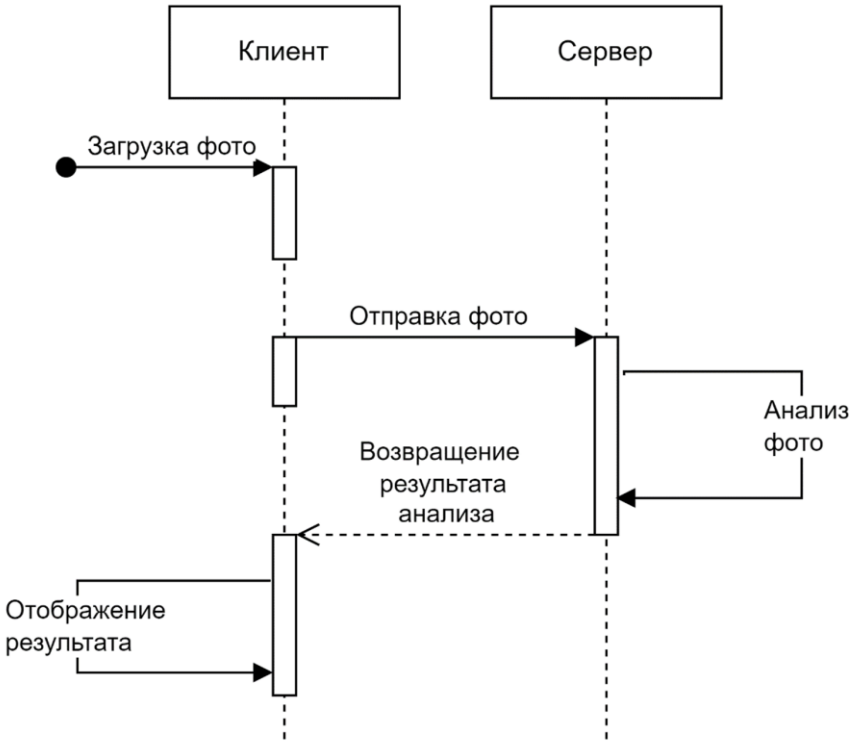


Рисунок 12 – Диаграмма последовательности

Перед разработкой сервиса необходимо спроектировать структуру проекта. Общая структура проекта представлена в виде UML-диаграммы на рисунке 13.

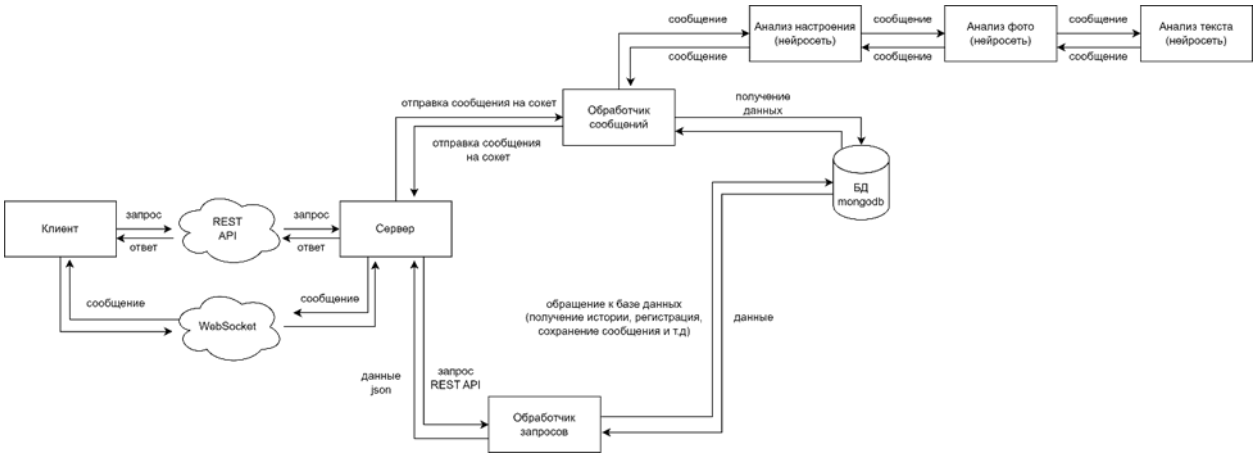


Рисунок 13 – Общая структура проекта

Клиент взаимодействует с сервером, отправляя серверу запросы. Если пользователь осуществляет взаимодействие с чат-ботом или системой, диагностирующей заболевания по фотографии, то отправленные данные обрабатываются нейронной сетью. Нейронная сеть обрабатывает полученные

от пользователя данные. После обработки данных пользователя, система отправляет результат клиенту. При получении результата первичной диагностики, пользователю будут выведены рекомендации для лечения заболевания или/и рекомендации по оказанию первой медицинской помощи.

Была спроектирована *UML*-диаграмма развёртывания. Диаграмма представлена на рисунке 14.

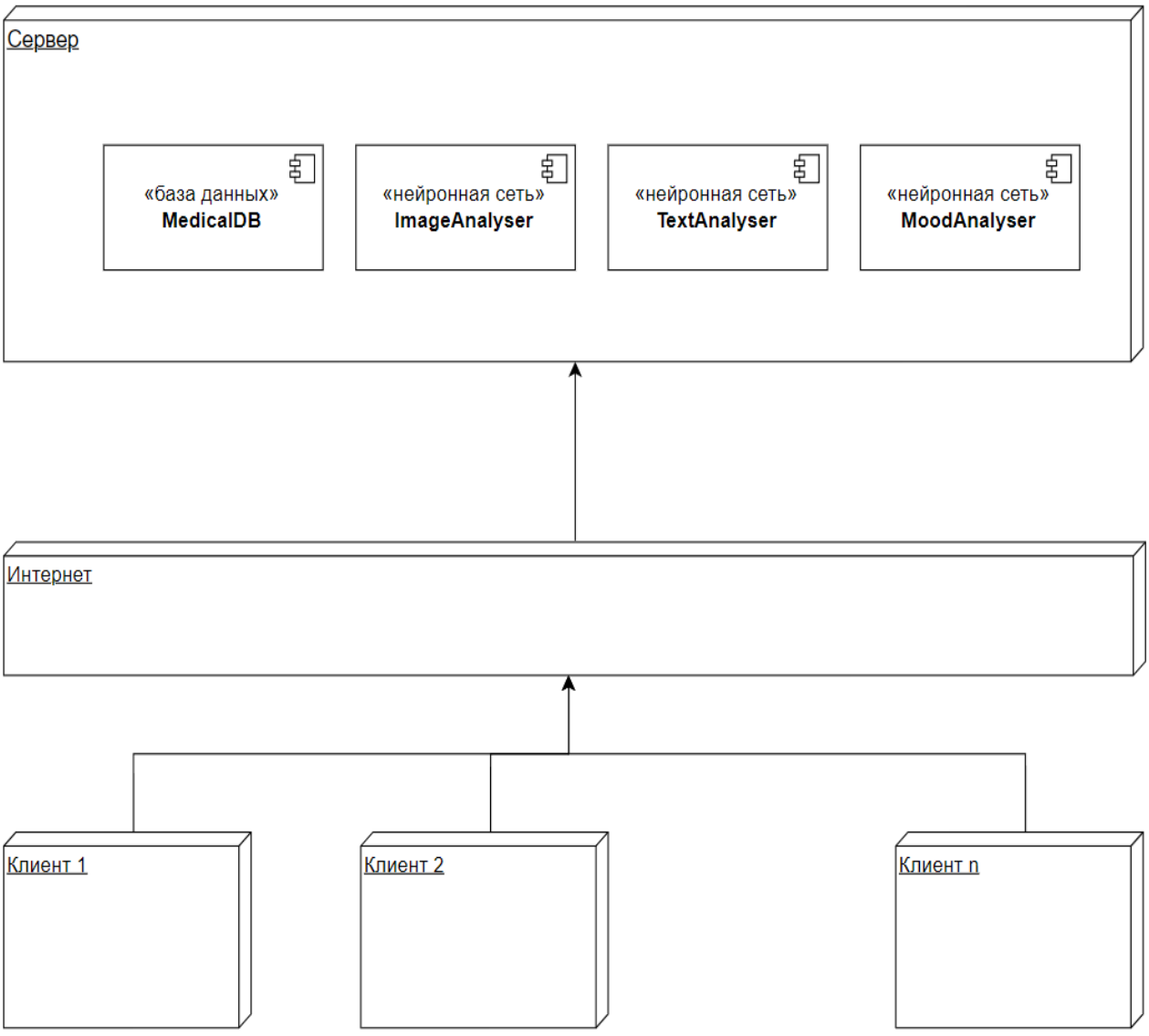


Рисунок 14 - Диаграмма развертывания

После того, как была спроектирована диаграмма развертывания, была разработана диаграмма прецендентов. Диаграмма прецендентов необходима для определения функционала пользователя. Диаграмма представлена на рисунке 15.

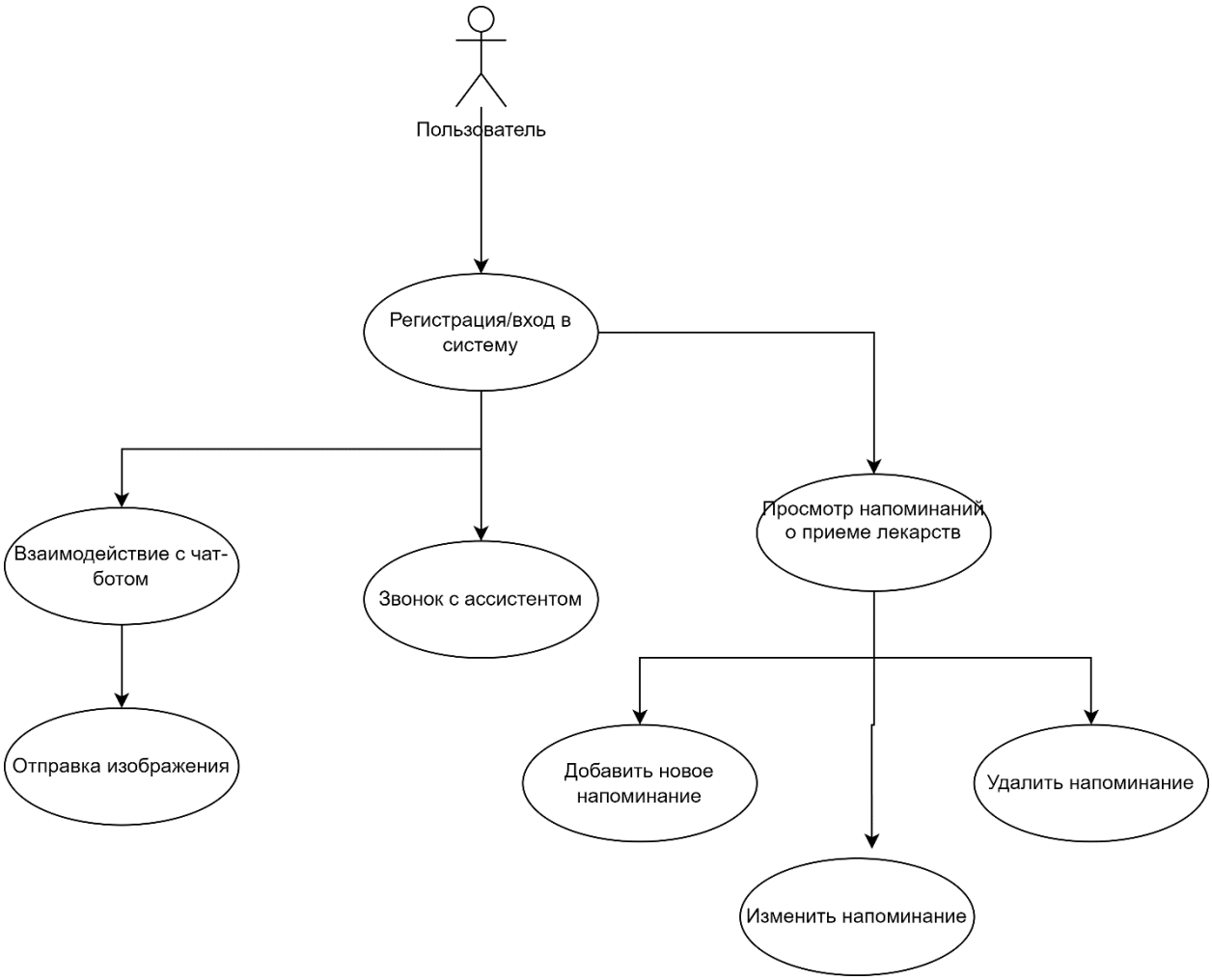


Рисунок 15 – Диаграмма прецедентов

1.8 Выбор языка программирования и среды разработки для клиентского приложения

В данный момент самыми популярными операционными системами для мобильных телефонов являются iOS и Android. По статистике доля пользователей Android составляет 71% от общего количества пользователей мобильными устройствами, а доля пользователей iOS – 28%. Исходя из этой статистики было принято решение разрабатывать мобильное приложение для двух самых распространённых операционных систем – Android и iOS.

Есть два наиболее популярных фреймворка для создания кроссплатформенных мобильных приложений – Flutter и React Native. Рассмотрим каждый по отдельности.

Flutter – фреймворк для создания кроссплатформенных приложений, разработанный и поддерживаемый компанией Google. Во Flutter для написания приложений используется язык Dart. Это объектно-ориентированный язык, созданный в качестве замены JavaScript. Язык имеет строгую статическую типизацию и простой современный синтаксис. Рендеринг во Flutter выполняется с помощью своего собственного движка – Skia [2], что обеспечивает высокую производительность.

React Native – фреймворк для разработки кроссплатформенных приложений, созданный компанией Facebook в 2015 году. В качестве языка программирования приложений используется JavaScript.

Сравнительная таблица представлена в таблице 2 ниже.

Таблица 2 – Сравнение фреймворков для написания кроссплатформенного мобильного приложения

Программа	Достоинства	Недостатки
Flutter	Собственный движок рендеринга Skia обеспечивает высокую производительность при отрисовке сложных элементов или анимации, что является одним из ключевых преимуществ фреймворка Flutter.	В качестве языка программирования используется Dart, что означает, что проект будет написан на этом языке, исключая возможность использования других языков программирования.
React Native	В качестве языка программирования используется JavaScript, что позволяет написать проект целиком на этом языке, что может быть удобно для проектов, которые уже используют JavaScript в веб-версии.	Использование моста для вызова нативных функций может привести к проблемам с производительностью, а также требует значительного количества ресурсов центрального процессора.

Проанализировав все преимущества и недостатки каждой технологии, а также учитывая поставленные задачи на разработку, было принято решение разрабатывать мобильное приложение с использованием фреймворка Flutter.

1.9 Выбор фреймворка машинного обучения

Tensor Flow - фреймворк с открытым исходным кодом для программирования для разработки нейронных сетей. Фреймворк сложен в использовании.

Keras - фреймворк нейронных сетей с открытым исходным кодом, написанный на Python. Keras обладает относительно низкой производительностью, из-за чего фреймворк желательно использовать только для небольших наборов данных. Эти особенности Keras могут стать существенными недостатками при дальнейшей разработке и возможного расширения функционала программы.

PyTorch - фреймворк машинного обучения с открытым исходным кодом для Python, основанная на torch. Фремворк был разработан исследовательской группой Facebook по искусственному интеллекту. PyTorch обладает удобной отладкой программы, удобно использовать для высокопроизводительных моделей и больших наборов данных, требующих высокой скорости выполнения. На официальном сайте PyTorch есть огромное количество обучающих уроков для новичков, готовых моделей нейронных сетей.

Основные особенности PyTorch:

– отлично подходит для разработки решений связанных с классификацией изображений, детектированием и сегментацией,

– существует большое количество готовых моделей, обученных на большом массиве данных, которые можно легко использовать для решения различных задач,

– используются динамические вычислительные графы, которые не требуют компиляции перед каждым выполнением. Данные можно изменять прямо во время работы.

Для разрабатываемого приложения самым подходящим фреймворком является PyTorch.

1.10 Выбор языка программирования и среды разработки для серверного приложения

Выбор языка программирования зависит от вышеописанного фреймворка машинного обучения. Именно поэтому был выбран язык Python. Также для удобства и скорости разработки, сервер будет также написан с помощью этого языка.

В качестве фреймворка для создания сервера был выбран Flask. Он очень легковесный и позволяет в кратчайшие сроки описать запросы для REST API. Является достаточно простым фреймворком, но несмотря на это, он достаточно функциональный, что позволяет его использовать в совершенно разных проектах. Также при столкновении с проблемой или трудностью при разработке, всегда можно обратиться к большому сообществу разработчиков.

1.11 Выводы

Таким образом, в данном разделе был проведен анализ темы и аналогичных проектов, была определена архитектура проекта, а также была определен алгоритм, по которому будет происходит взаимодействие с общей системой.

2 Разработка модуля первичной диагностики для мобильного приложения

2.1 Разработка серверной части для модуля первичной диагностики

2.1.1 Разработка серверного приложения

После успешного обучения нейронной сети, нужно обеспечить к ней доступ через REST API.

Для этого был использован фреймворк Flask, который позволяет создавать серверные приложения в кратчайшие сроки.

Изначально необходимо описать всю логику для анализа изображения. Создадим отдельный файл и опишем в нем функцию detect_disease.

В эту функцию передается путь к картинке, которую нужно проанализировать, далее выполняется преобразование в тензор для пропуска через нейронную сеть.

Определяется количество классов, далее, если доступны ресурсы видеокарты для вычисления, используем ее, в противном случае вычисления будут выполняться на cpu, однако будут значительно медленнее.

Далее выполняем загрузку модели используя файл с расширением pth, и выполняем анализ с помощью нейронной сети.

В конце происходит формирование результата, который будет отправлен клиенту.

Код представлен на рисунке 16 ниже.

```python
def detect_disease(img_path):
    img = Image.open(img_path).convert("RGB")
    img = F.to_tensor(img)

    num_classes = NUM_CLASSES
    model = get_object_detection_model(num_classes)
    device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
    model.to(device)
    model.load_state_dict(torch.load('models/detector.pth', map_location=device))

    model.eval()
    with torch.no_grad():
      prediction = model([img.to(device)])[0]

    nms_prediction = apply_nms(prediction, iou_thresh=0.01)
    for key in nms_prediction.keys():
      nms_prediction[key] = nms_prediction[key].tolist()

    new_boxes = []
    for boxes in nms_prediction['boxes']:
      new_boxes.append({'box': boxes})

    new_labels = []
    for label in nms_prediction['labels']:
        new_labels.append(disease_list[label])

    nms_prediction['boxes'] = new_boxes
    nms_prediction['labels'] = new_labels

    result = []
    for ind, label in enumerate(new_labels):
        pred = {"label": label, "box": new_boxes[ind]['box'], "score": nms_prediction['scores'][ind]}
        result.append(pred)

    return result
```

Рисунок 16 – Выполнение анализа

Загрузка модели происходит с помощью функции get_object_detection_model, которая представлена на рисунке 17 ниже.

```python
def get_object_detection_model(num_classes):
    model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)
    in_features = model.roi_heads.box_predictor.cls_score.in_features
    model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)
    return model
```

Рисунок 17 – Загрузка модели нейронной сети

Функция apply_nms применяется, чтобы отбросить все ограничивающие рамки, которые имеют пересечение друг с другом. Код функции представлен на рисунке 18 ниже.

```
def apply_nms(orig_prediction, iou_thresh=0.3):
    keep = torchvision.ops.nms(orig_prediction['boxes'], orig_prediction['scores'], iou_thresh)

    final_prediction = orig_prediction
    final_prediction['boxes'] = final_prediction['boxes'][keep]
    final_prediction['scores'] = final_prediction['scores'][keep]
    final_prediction['labels'] = final_prediction['labels'][keep]

    return final_prediction
```

Рисунок 18 – Фильтрация ограничивающих рамок

Так как модель нейронной сети обладает достаточно большим объемом, занимающим место на диске, было принято решение разработать также запрос для загрузки модели на выделенный сервер. Он также в будущем сможет использоваться в панели администратора для управления нейронной сетью и замены модели на более совершенную.

Первый запрос служит для выполнения анализа фотографии от пользователя. Изображение, которое передается в запросе, сохраняется в временный файл temp.png и в дальнейшем используется для анализа нейронной сетью с помощью функции detect_disease.

На выходе получается объект, состоящий из ограничивающей рамки, названия, а также вероятности наличия заболевания.

На рисунке 19 ниже представлен запрос для выполнения анализа изображения.

```python
@app.route("/image-analysis", methods = ["GET"])
def analyze_image():

    imageBase64 = request.json.get('image')
    print(imageBase64)
    with open("temp.png", "wb") as f:
        f.write(base64.b64decode(imageBase64))
        res = detector.detect_disease("temp.png")
        return jsonify({"result": res})
```

Рисунок 19 – Запрос для выполнения анализа изображения

Второй запрос служит для загрузки модели нейронной сети на сервер.

Модель берется из запроса (request.files) и сохраняется с расширением pth. Клиенту возвращается результат операции.

Запрос представлен на рисунке 20 ниже.

```python
@app.route("/image-analysis-network", methods = ["POST"])
def upload_image_nn():

    uploaded_file = request.files["file"]
    ext = uploaded_file.filename.split('.')[1]
    if uploaded_file.filename != '' and ext == 'pth':
        uploaded_file.save("models/detector.pth")
        return jsonify({"success": True})
    return jsonify({"success": False})
```

Рисунок 20 – Ответ для клиента

Запуск сервера происходит с помощью Flask, код представлен на рисунке 21 ниже.

```python
if __name__ == '__main__':
    app.run(port=3001)
```

Рисунок 21 – Запуск сервера

2.1.2 Развертывание серверного приложения

Развертывание серверного приложения — это важный процесс, который включает в себя установку, настройку и готовку приложения к работе на сервере, чтобы оно было доступно для пользователей.

Эффективное развертывание обеспечивает стабильную работу приложения, удобство обновлений и масштабирование системы в соответствии с потребностями пользователей и бизнеса. Для упрощения процесса развертывания серверного приложения будет использован Docker - платформа для разработки, доставки и запуска приложений в контейнерах. Использование Docker позволяет создавать контейнеры, которые могут быть легко переносимы между различными средами, что упрощает развертывание приложения на разных серверах или облачных платформах. Для создания выделенного сервера будет использоваться специальный сервис, который предоставляет доступ к серверам за определенную плату.

2.1.3 Контейнеризация приложения

Контейнеризация приложения — это способ упаковки и запуска приложения вместе со всеми его зависимостями (библиотеки, настройки, код) в изолированном окружении, называемом контейнером.

Основными преимуществами контейнеризации являются:

– контейнер работает одинаково на любой платформе не зависимо от окружения,

– каждый сервис запускается в изолированном контейнере, это предотвращает конфликты между зависимостями разных сервисов,

– легко масштабируются, контейнеры можно быстро создавать, останавливать и перемещать,

– занимают меньше места и меньше ресурсов, чем виртуальные машины.

Исходя из вышеперечисленного можно сделать вывод, что контейнеризация делает разработку, тестирование и развертывание приложений быстрым и надежным.

2.1.4 Docker

Докер (Docker) — это программная платформа для быстрой сборки, отладки и развертывания приложений с помощью контейнеров. Контейнеры

— это изолированные окружения, которые содержат все необходимые для приложения зависимости, такие как код, библиотеки и настройки. Докер позволяет создавать контейнеры, которые могут запускаться на любом устройстве, где установлен Docker, без необходимости настройки окружения вручную.

Докер работает следующим образом:

– докер создает образ приложения, который содержит все необходимые зависимости,

– докер создает контейнер на основе образа и запускает приложение,

– контейнер работает как отдельное окружение, не влияя на другие приложения на устройстве,

– докер позволяет управлять контейнером, останавливать, запускать и масштабировать его.

Докер упрощает процесс развертывания приложений, обеспечивая изоляцию и масштабируемость. Это делает его популярным инструментом для разработчиков и операторов приложений

В корне проекта создадим файл Dockerfile. И впишем следующие инструкции, представленные ниже.

```
FROM python:3.11
WORKDIR /usr/src/app
COPY ./requirements.txt /usr/src/app/requirements.txt
RUN pip install -r requirements.txt
COPY . .
ENTRYPOINT ["python"]
CMD ["app.py"]
```

Этот Dockerfile описывает процесс создания Docker-образа для Python-приложения.

Базовый образ: `FROM python:3.11`

Этот оператор указывает, что в качестве базового образа будет использоваться официальный образ Python версии 3.11.

Рабочая директория: `WORKDIR /usr/src/app`

Этот оператор устанавливает рабочую директорию внутри контейнера в /usr/src/app. Все последующие команды будут выполняться относительно этой директории.

Копирование `requirements.txt`: `COPY./requirements.txt/usr/src/app/requirements.txt`

Этот оператор копирует файл requirements.txt из текущей директории на хосте в рабочую директорию /usr/src/app внутри контейнера. Этот файл содержит список зависимостей Python-приложения.

Установка зависимостей: `RUN pip install -r requirements.txt`

Этот оператор запускает команду pip install внутри контейнера, устанавливая все зависимости, перечисленные в файле requirements.txt.

Копирование приложения: `COPY . .`

Этот оператор копирует все файлы и директории из текущей директории на хосте в рабочую директорию /usr/src/app внутри контейнера.

Точка входа: `ENTRYPOINT ["python"]`

Этот оператор устанавливает точку входа в контейнер - команду python, которая будет выполняться при запуске контейнера.

Команда по умолчанию: `CMD ["app.py"]`

Этот оператор устанавливает команду, которая будет выполняться по умолчанию при запуске контейнера. В данном случае это будет файл app.py.

Таким образом, этот Dockerfile описывает процесс создания Docker-образа для Python-приложения. Он использует базовый образ Python 3.11, устанавливает необходимые зависимости, копирует приложение в контейнер и задает точку входа и команду по умолчанию.

2.1.5 Docker-compose

Docker Compose - это инструмент, который позволяет определять и запускать многоконтейнерные приложения с помощью файла конфигурации YAML. Он упрощает процесс развертывания и управления приложениями,

позволяя одновременно определять несколько контейнеров и их взаимодействие. Для описания структуры многоконтейнерного приложения необходимо использовать синтаксис YAML. В целом, Docker Compose является очень мощным инструментом для разработчиков и DevOps-специалистов, который позволяет существенно упростить процесс работы с контейнеризированными приложениями.

В корне проекта необходимо создать файл docker-compose с расширением yml. Далее необходимо определить контейнеры, которые будут использоваться в нашем приложении, включая базу данных. В качестве образа укажем последнюю актуальную версию.

```
version: "3.4"
services:
  medical-app:
    image: python:3.11
    build:
      context: .
    volumes:
      - ./storage:/usr/src/app/storage
    ports:
      - "3001:3001"
    restart: always
    networks:
      - my-network

networks:
  my-network:
    driver: bridge
```

Этот код Docker Compose описывает создание и настройку сервиса medical-app и сети my-network.

Версия Docker Compose: version: "3.4"

Эта строка указывает, что используется версия Docker Compose 3.4.

Сервис medical-app:

```
services:
  medical-app:
    image: python:3.11
    build:
      context: .
    volumes:
      - ./storage:/usr/src/app/storage
    ports:
      - "3001:3001"
    restart: always
    networks:
      - my-network
```

Описание сервиса medical-app:

– image указывает, что сервис будет использовать образ python:3.11 для запуска.

– build указывает, что сервис будет собран из текущей директории (.).

– volumes указывает, что директория ./storage на хосте будет монтирована в /usr/src/app/storage внутри контейнера. Это позволяет хранить данные в директории storage на хосте и доступ к ним из контейнера.

– ports указывает, что порт 3001 на хосте будет маппирован на порт 3001 внутри контейнера. Это позволяет доступ к приложению из внешней сети.

– restart указывает, что сервис будет автоматически перезапущен, если он остановится.

– networks указывает, что сервис будет подключен к сети my-network.

Сеть my-network:

```
networks:
  my-network:
    driver: bridge
```

Описание сети my-network: driver: указывает, что сеть будет создаваться с помощью драйвера bridge. Это позволяет создавать сети, которые могут быть доступны из внешней сети.

В целом, этот код создает сервис medical-app, который использует образ python:3.11, монтирует директорию ./storage на хосте в контейнер, доступен по порту 3001 и автоматически перезапускается, если остановится. Он также подключен к сети my-network, создаваемой с помощью драйвера bridge.

Приложение предоставляет два основных функционала: загрузку моделей нейронной сети и анализ изображений. Для реализации этих возможностей в коде определены два маршрута: /image-analysis-network и /image-analysis.

POST: /image-analysis-network - метод используется для загрузки моделей нейронной сети. Метод проверяет аутентификацию пользователя с помощью функции check_auth(). Извлекает загруженный файл из запроса и сохраняет его как detector.pth. Возвращает ответ JSON, указывающий, был ли файл успешно загружен или нет.

POST /image-analysis – метод используется для анализа изображений. Проверяет аутентификацию пользователя с помощью функции check_auth(). Извлекает кодированное в base64 изображение из запроса и декодирует его. Вызывает функцию detector.check_herpes() для анализа изображения и возвращает результат в ответе JSON.

Код создает веб-приложение с двумя маршрутами для загрузки моделей нейронной сети и анализа изображений. Проверяет аутентификацию

пользователя для защиты доступа к приложению. Загружает файлы и анализирует изображения с помощью функции detector.check_herpes().

2.1.6 GIT/GITHUB и разворачивание приложения

Git и GitHub — это два важных инструмента для разработчиков программного обеспечения, которые обеспечивают эффективное управление версиями кода и совместную работу над проектами. Git — это система контроля версий, разработанная Линусом Торвальдсом, которая позволяет разработчикам отслеживать изменения в исходном коде, создавать ветви для параллельной разработки и отслеживать историю изменений проекта. GitHub — это сервис для хостинга репозиториев Git, который обеспечивает возможность хранения, совместной работы и отслеживания изменений в проектах.

Для загрузки репозитория с GitHub на удаленном сервере выполним команду git clone git@github.com:$repository_ssh. Поскольку Docker уже настроен и создан файл конфигурации docker-compose.yml, можно выполнить команду docker-compose up --build для запуска приложения. После выполнения этих шагов сервер будет готов к работе и может быть использован клиентами.

2.2 Разработка клиентской части

Модули клиентского приложения были разработаны с помощью фреймворка Flutter на языке Dart.

Сюда входят следующие части системы:

– первичная диагностика заболеваний с помощью изображений,

– звонок с виртуальным ассистентом,

– добавление и настройка уведомлений для приема лекарств.

Рассмотрим каждую часть по очереди.

Прием лекарств. В профиле пользователя был создан раздел, который будет перенаправлять на отдельную страницу, на которой будет

отображаться список со всеми настроенными уведомлениями. Профиль пользователя показан на рисунке 22 ниже.

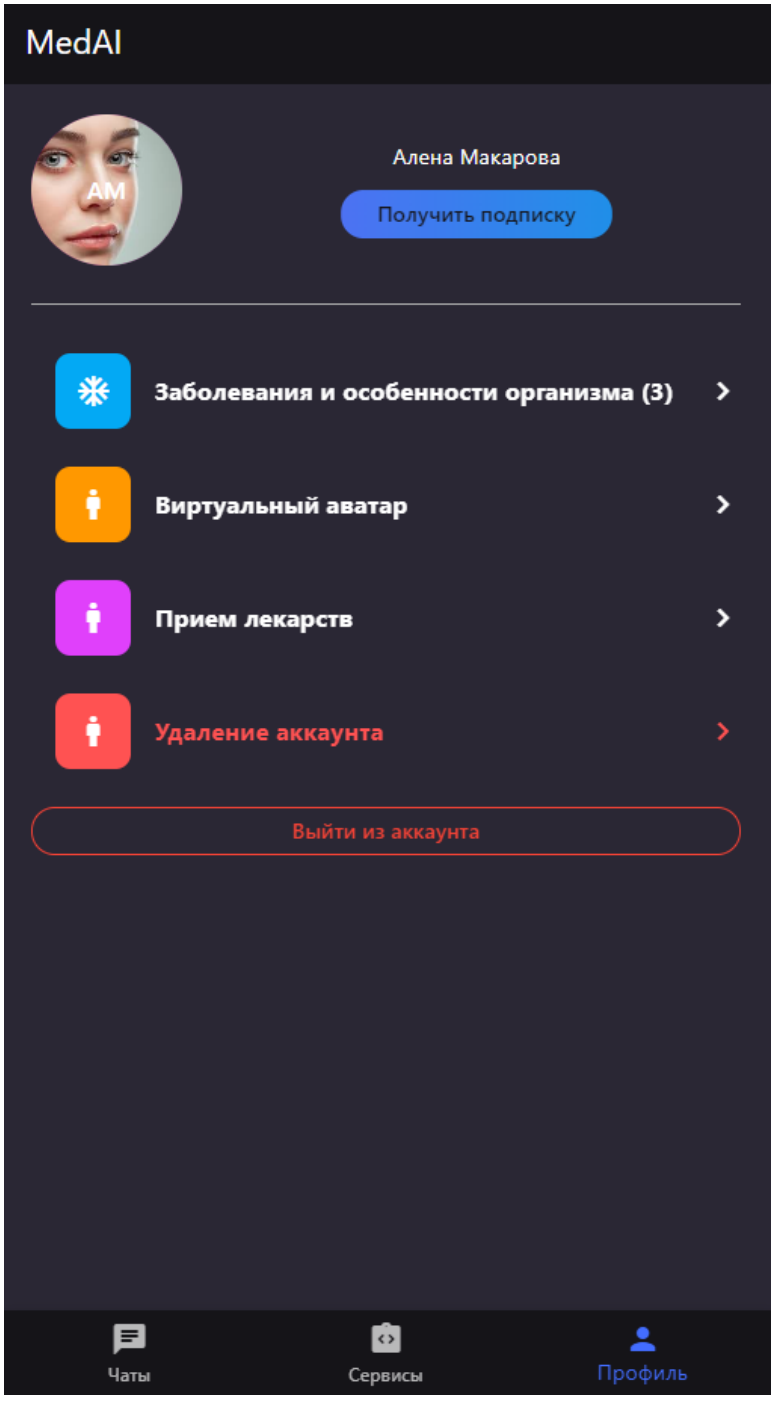


Рисунок 22 – Страница «Профиль пользователя»

При нажатии на пункт «Прием лекарств» будет выполнено перенаправление на соответствующую страницу. На этом экране представлен весь список с настроенными уведомлениями, для каждого элемента отображается время, на которое назначен прием, а также дополнительная

информация о самом препарате, типе приема и др. Также на этом экране отображается кнопка в правом нижнем углу, с помощью которой можно настроить новые уведомление. При долгом нажатии на элемент на экране появляется меню, через которое можно выполнить удаление уведомления.

Интерфейс представлен на рисунке 23 ниже.

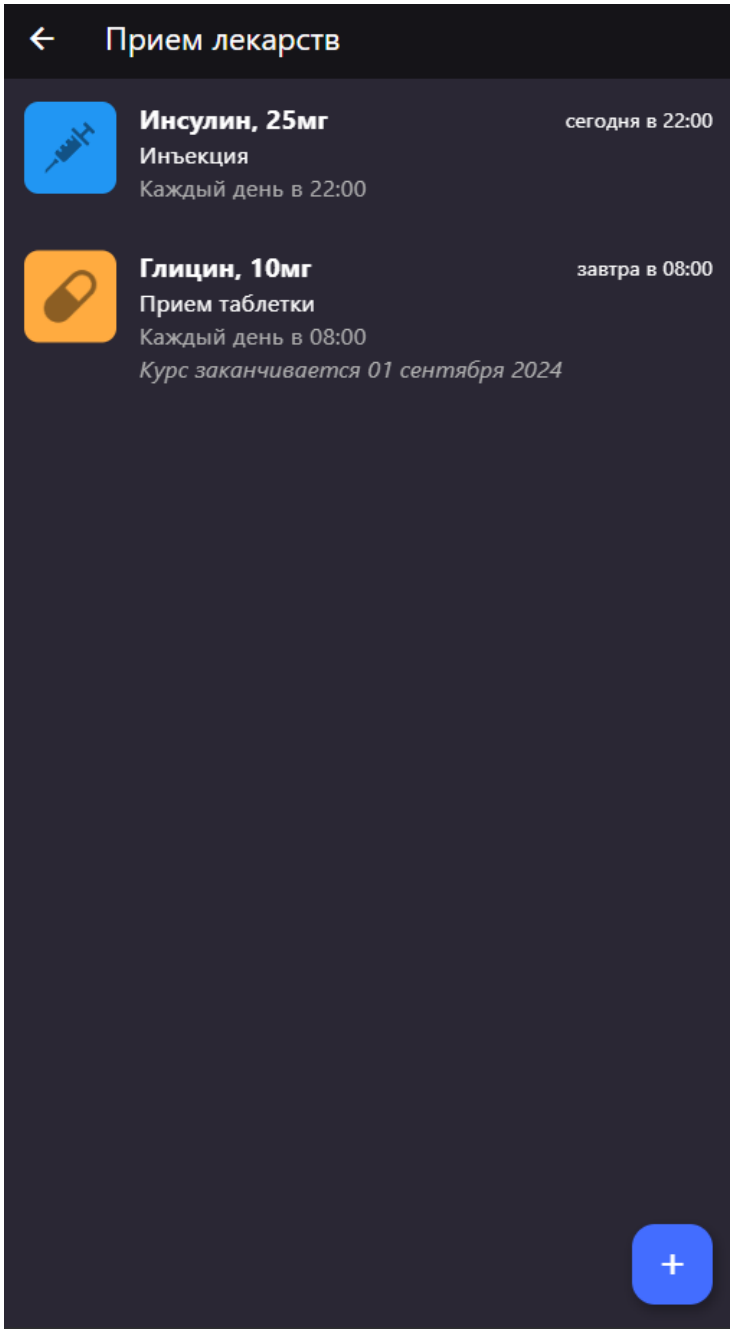


Рисунок 23– Страница «Прием лекарств»

Создание нового уведомления. На этом экране может выполняться настройка для создания уведомления, можно указать тип приема, название,

настроить время. После ввода всех данных можно будет сохранить уведомление. Введенные данные дублируются в базе данных, а также сохраняются на сервере для возможной обработки на других клиентах в будущем. После сохранения новой записи, она появится в общем списке с препаратами. А по достижению назначенного времени у пользователя отобразится уведомление, при нажатии на которое можно будет отметить прием. Данные занесутся в базу, а также обновятся на сервере. В будущем можно будет отобразить полноценную таблицу, чтобы отслеживать все изменения, а также использовать для более точных рекомендаций. Такие данные также может использовать врач при наличии интеграции с сетями медицинских клиник. Таким образом, врач сможет видеть, принимает ли пациент назначенные ему препараты.

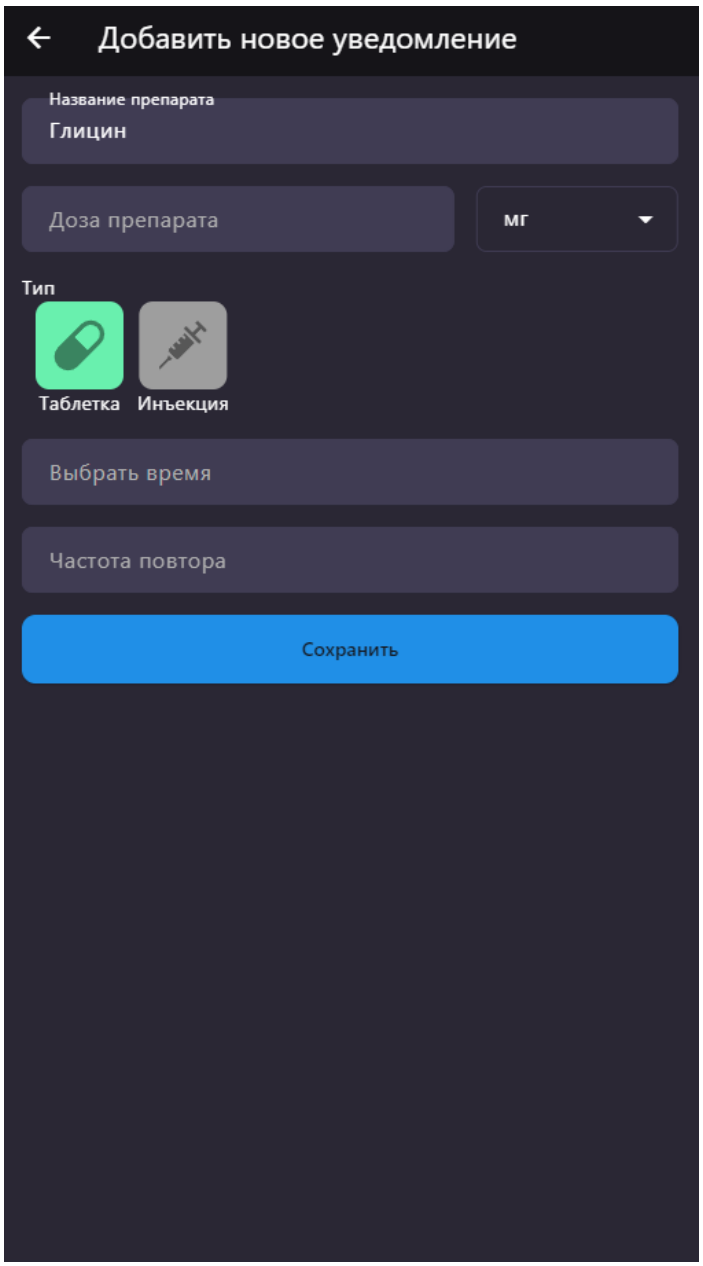


Рисунок 24 – Страница «Добавить новое уведомление»

Звонок с виртуальным ассистентом. В чате боте, связанных с текстовым анализом, в верхней панели отображается трубка, которая позволяет начать звонок.

Интерфейс чата представлен на рисунке 25.

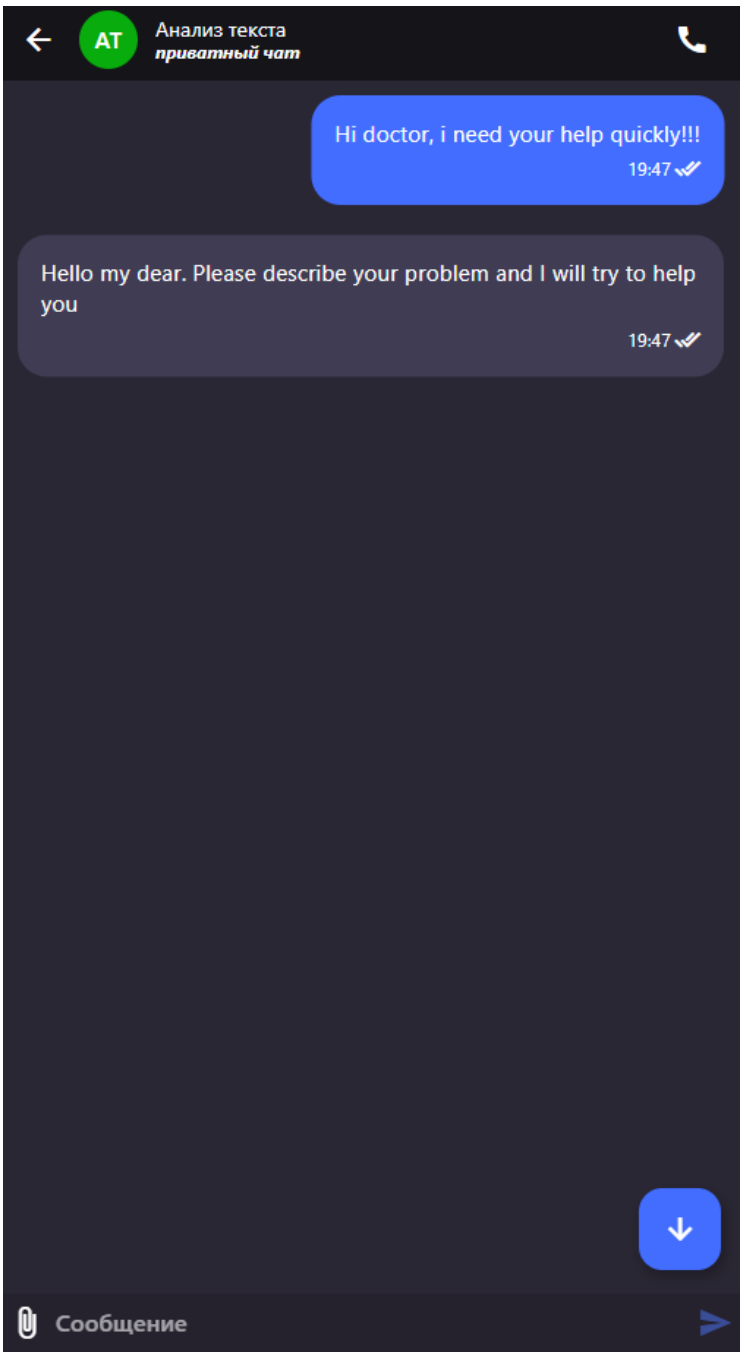


Рисунок 25 – Страница «Чат-бот»

После нажатия на трубку выполнится передох на другой экран, который представляет собой звонок. Для выполнения самого звонка были использованы библиотеки, которые выполняют преобразование текста в голос, и наоборот. К таким библиотекам относится fluter_tts и speech_to_text После того, как пользователь произнес свою фразу, она преобразуется в текстовые данные. И эти данные уже используются для выполнения запроса на сервер, который позволяет выполнить обработку. Результат отправляется

на сокет, чтобы клиент смог выполнить обработку. На клиенте же, в свою очередь, выполняется обратное преобразование – текст результата запроса переводится в голос и воспроизводится.

Следует отметить, что для корректной и эффективной работы такого функционала, требуются большие вычислительные мощности, в частности, необходимы мощные графический ускорители для того, чтобы модель нейронной сети смогла обрабатывать запросы за минимальное отведенное время и отправлять их на сокет.

Интерфейс представлен на рисунке 26 ниже.

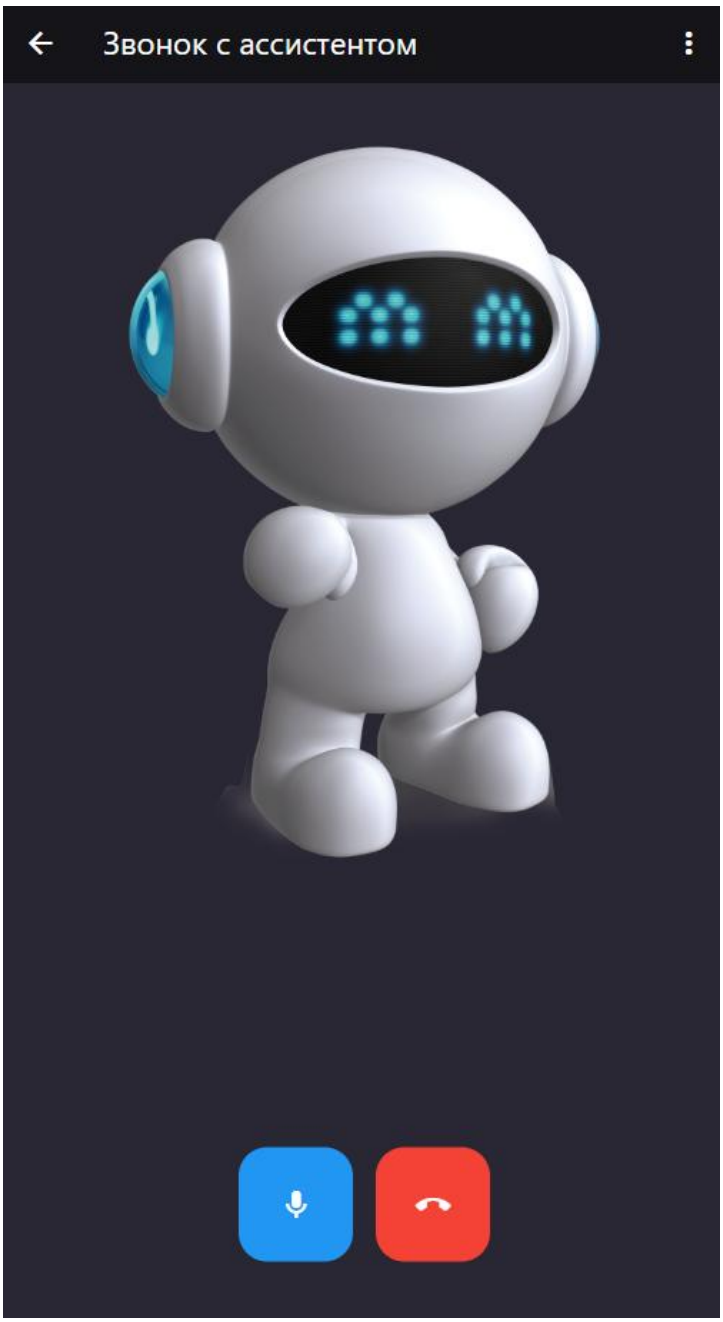


Рисунок 26 – Страница «Звонок с ассистентом»

После того, как пользователь захочет завершить звонок, он может нажать на красную трубку. Также на этом экране возможно сделать запись звонка, все вопросы пользователя и ответы от помощника будут записываться в базу данных и в дальнейшем отображаться в чате. Данная функция реализована на сервере.

2.3 Разработка нейронной сети

2.3.1 Описание классов

Дата сет содержит следующие классы:

– угревая сыпь и розацеа (Acne and Rosacea Photos),

– базалиома (Actinic Keratosis Basal Cell Carcinoma and other Malignant Lesions),

– атопический дерматит (Atopic Dermatitis Photos),

– буллезная болезнь (Bullous Disease Photos),

– инфекционный целлюлит (Cellulitis Impetigo and other Bacterial Infections),

– экзема (Eczema Photos),

– экзантема (Exanthems and Drug Eruptions),

– алопеция (Hair Loss Photos Alopecia and other Hair Diseases),

– герпес (Herpes HPV and other STDs Photos),

– гиперпигментация (Light Diseases and Disorders of Pigmentation),

– волчанка (Lupus and other Connective Tissue diseases),

– меланома (Melanoma Skin Cancer Nevi and Moles),

– онихомикоз (Nail Fungus and other Nail Disease),

– ядовитый плющ (Poison Ivy Photos and other Contact Dermatitis),

– красный плоский лишай (Psoriasis pictures Lichen Planus and related diseases),

– болезнь лайма (Scabies Lyme Disease and other Infestations and Bites),

– себорейный кератоз (Seborrheic Keratoses and other Benign Tumors),

– системное заболевание кожи (Systemic Disease),

– кандидоз (Tinea Ringworm Candidiasis and other Fungal Infections),

– аллергическая крапивница (Urticaria Hives),

– ангиома (Vascular Tumors),

– васкулит (Vasculitis Photos),

– контагиозный моллюск (Warts Molluscum and other Viral Infections).

2.3.2 Описание нейронной сети

Для начала необходимо привести изображения к единому формату и тензорам, которые необходимы для работы нейронной сети. Код представлен ниже:

```
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'test': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}
```

2.3.3 Загрузка данных

Далее необходимо определить каталог файлов откуда будем брать тренировочные изображения и изображения для тестов.

```
data_dir = 'data/'
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
                                          data_transforms[x])
                  for x in ['train', 'test']}
dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=4,
```

```
                                                shuffle=True,
num_workers=4)
              for x in ['train', 'test']}
dataset_sizes = {x: len(image_datasets[x]) for x in ['train',
'test']}
class_names = image_datasets['train'].classes
```

2.3.4 Отображение изображений

Код для отображения изображений представлен ниже:

```
def imshow(inp, title=None):
    """Display image for Tensor."""
    inp = inp.numpy().transpose((1, 2, 0))
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    inp = std * inp + mean
    inp = np.clip(inp, 0, 1)
    plt.imshow(inp)
    if title is not None:
        plt.title(title)
    plt.pause(0.001)  # pause a bit so that plots are updated
```

Получение пакета обучающих данных:

```
inputs, classes = next(iter(dataloaders['train']))
```

Необходимо сделать таблицу, для структурированного отображения данных:

```
out = torchvision.utils.make_grid(inputs)
imshow(out, title=[class_names[x] for x in classes])
```

В результате была получена таблица изображений с соответствующими именами классов. Результат представлен на рисунке 27:

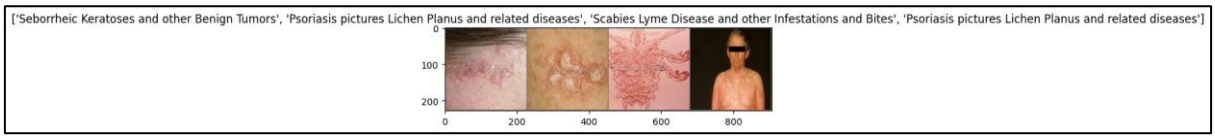


Рисунок 27 – Таблица изображений

2.3.5 Обучение модели

После форматирования изображений необходимо написать обучающую функцию. Эта функция продемонстрирует два важных аспекта:

– планирование скорости обучения,

– сохранение лучшей модели.

Функция будет использоваться для эффективного обучения модели [4]. Код представлен ниже:

```
def train_model(model, criterion, optimizer, scheduler, num_epochs=25):
    since = time.time()
```

Создание временного каталога для сохранения контрольных точек обучения:

```
    with TemporaryDirectory() as tempdir:
        best_model_params_path = os.path.join(tempdir, 'best_model_params.pt')

        torch.save(model.state_dict(), best_model_params_path)
        best_acc = 0.0

        for epoch in range(num_epochs):
            print(f'Epoch {epoch}/{num_epochs - 1}')
            print('-' * 10)
```

Каждая эпоха имеет фазу обучения и проверки:

```
            for phase in ['train', 'test']:
                if phase == 'train':
```

```
                model.train()  # Set model to training mode
            else:
                model.eval()   # Set model to evaluate mode

            running_loss = 0.0
            running_corrects = 0
```

Выполнение итерации по данным:

```
            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)
```

Приведение к нулевому значению градиентов параметра:

```
optimizer.zero_grad()
with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
    loss = criterion(outputs, labels)
```

Обратная оптимизация:

```
                        if phase == 'train':
                        loss.backward()
                        optimizer.step()
```

Статистика результатов обучения:

```
                running_loss += loss.item() * inputs.size(0)
                running_corrects   +=   torch.sum(preds   ==
labels.data)
if phase == 'train':
                scheduler.step()
epoch_loss = running_loss / dataset_sizes[phase]
```

```
epoch_acc = running_corrects.double() / dataset_sizes[phase]


print(f'{phase} Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}')
 if phase == 'test' and epoch_acc > best_acc:
                    best_acc = epoch_acc
                    torch.save(model.state_dict(),
best_model_params_path)
            print()
        time_elapsed = time.time() - since
        print(f'Training complete in {time_elapsed // 60:.0f}m
{time_elapsed % 60:.0f}s')
         print(f'Best val Acc: {best_acc:4f}')
```

Загрузка лучшего веса модели: `model.load_state_dict(torch.load(best_model_params_path))`

```
    return model
```

2.3.6 Функция для визуализации прогнозов модели

Для визуализации прогнозов модели на вводных изображениях необходимо сделать функцию для визуализации изображений. Код представлен ниже:

```
def visualize_model(model, num_images=6):
    was_training = model.training
    model.eval()
    images_so_far = 0
    fig = plt.figure()


    with torch.no_grad():
                        for     i,     (inputs,     labels)     in
enumerate(dataloaders['test']):
            inputs = inputs.to(device)
            labels = labels.to(device)
```

```
            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)

            for j in range(inputs.size()[0]):
                images_so_far += 1
                            ax  =  plt.subplot(num_images//2,  2,
images_so_far)
                ax.axis('off')
                                        ax.set_title(f'predicted:
{class_names[preds[j]]}')
                imshow(inputs.cpu().data[j])

                if images_so_far == num_images:
                    model.train(mode=was_training)
                    return
        model.train(mode=was_training)
```

2.3.7 Настройка ConvNet

Для быстрого обучения модель необходимо использовать предварительно обученную модель. Для определения заболеваний по изображениям лучше всего подходит resnet50. Код настройки предобученной модели представлен ниже:

```
model_ft = models.resnet50(weights='IMAGENET1K_V2')
num_ftrs = model_ft.fc.in_features
``nn.Linear(num_ftrs, len(class_names))``.
model_ft.fc = nn.Linear(num_ftrs, 23)
model_ft = model_ft.to(device)
criterion = nn.CrossEntropyLoss()
```

Оптимизация всех параметров:

```
optimizer_ft    =    optim.SGD(model_ft.parameters(),    lr=0.005,
momentum=0.9)
```

```
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft,
step_size=7, gamma=0.1)
```

Результат:

Downloading: "https://download.pytorch.org/models/resnet50-11ad3fa6.pth" to C:\Users\Rin/.cache\torch\hub\checkpoints\resnet50-11ad3fa6.pth

100%|██| 97.8M/97.8M [00:09<00:00, 11.2MB/s]

2.3.8 Обучение нейронной сети

Предобученная модель обучалась 50 эпох. Процесс обучения занял 564 минуты. В результате был получен следующий результат:

```
Epoch 0/49
----------
train Loss: 0.6725 Acc: 0.4079
test Loss: 0.5779 Acc: 0.4841

Epoch 1/49
----------
train Loss: 0.4517 Acc: 0.4662
test Loss: 0.5670 Acc: 0.5076

Epoch 2/49
----------
train Loss: 1.3224 Acc: 0.5082
test Loss: 2.6549 Acc: 0.5236

Epoch 3/49
----------
train Loss: 1.2439 Acc: 0.5252
```

```
test Loss: 1.5937 Acc: 0.5311

Epoch 4/49
----------
train Loss: 1.1551 Acc: 0.5537
test Loss: 1.8706 Acc: 0.5478
...
test Loss: 1.3587 Acc: 0.6818

Training complete in 564m 10s
Best val Acc: 0.708996
```

Точность нейронной сети после обучения составляет 70%.

2.3.9 Визуализация результатов

Вывод результатов по пользовательским изображениях представлен ниже. Для первого изображения результат представлен на рисунке 28, для второго на рисунке 29, для третьего на рисунке 30

Рисунок 28– Определение заболевания по изображению 1

Рисунок 29 – Определение заболевания по изображению 2

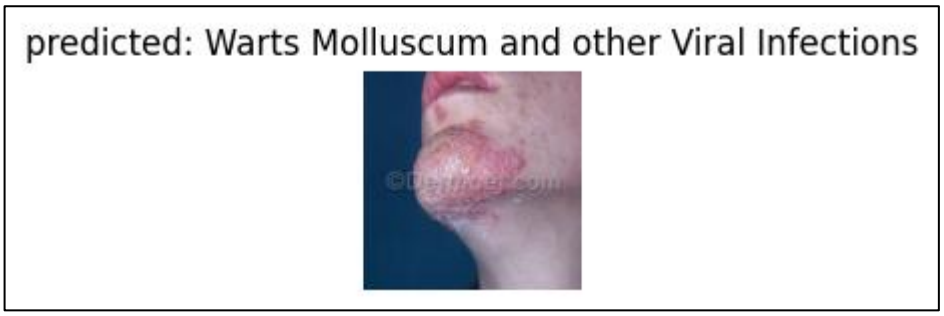


Рисунок 30 – Определение заболевания по изображению 3

2.4 Выводы

В результате была разработана нейронная сеть, которая позволяет анализировать фотографии пользователя и выдавать результат, который включает в себя ограничивающие рамки заболевания, название заболевания, а также вероятность его наличия.

Разработанная сеть была совмещена с сервером на Flask для взаимодействия с общей системой. Разработанная система позволит пользователям отправлять сообщение чат-боту и выполнять диагностику заболевания.

Сервер был развернут и полностью подготовлен для клиента.

ЗАКЛЮЧЕНИЕ

Проведен анализ темы и аналогичных приложений. После чего был определен и составлен план работ.

Выполнен сбор данных для обучения нейронной сети, выполнено обучение. Разработана нейронная сеть для определения заболеваний пользователей по изображениям. Нейронная сеть была внедрена в приложение на Flask и выведена на отдельный сервер.

Разработанная подсистема является частью общей комплексной системы, которая необходима для диагностики пользователя, путем общения с чат-ботом, который генерирует текст.

Реализованы запросы REST API, для загрузки модели нейронной сети и выполнения самого анализа. Загрузка модели может понадобиться в будущем, когда можно будет выполнить замену устаревшей модели на более современную и эффективную.

Разработанная подсистема может помочь пользователю следить за здоровьем, пользователь может настраивать уведомления для приема таблеток или применения инъекций и может существенно помочь человеку своевременно и быстро проверить собственный организм для диагностики. Система сможет сократить количество поздно диагностированных болезней.

В будущем планируется расширить функционал подсистемы, улучшить точность нейронной сети, увеличить количество классов заболеваний, которые будут определяться.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. LeCun, Y.: Gradient-Based Learning Applied to Document Recognition. Proceedings of the IEEE, 1998 г. – 46 с.

2. Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint arXiv:1409.1556, 2014.

3. He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 778 с.

4. Максимова Д. Р., Коротов В.О., Митрохин М.А - ПЕРВИЧНАЯ ДИАГНОСТИКА ЗАБОЛЕВАНИЙ ПО ФОТОГРАФИИ ПРИ ПОМОЩИ НЕЙРОННЫХ СЕТЕЙ // Сборник научных статей по материалам XIX Международной научно-технической конференции, посвященной 75-летию кафедры «Вычислительная техника» ПГУ. Пенза, 2022. Издательство: Пензенский государственный университет (Пенза). С. 303-305. URL: https://www.elibrary.ru/item.asp?id=50454563 (дата обращения: 07.06.2024).

ПРИЛОЖЕНИЕ А

(обязательное)

Медицинское приложение. Модуль первичной диагностики заболевания

Листинг программы

Листинг А.1 – Файл «Flask.py»

```
from flask import Flask, jsonify, request, abort
from waitress import serve
import analyzer.detection as detector
import base64
import requests
import os

app = Flask(__name__)

@app.route("/image-analysis-network", methods = ["POST"])
def upload_image_nn():
    if not check_auth(): abort(500, "Error")

    uploaded_file = request.files["file"]
    ext = uploaded_file.filename.split('.')[1]
    if uploaded_file.filename != '' and ext == 'pth':
        uploaded_file.save("detector.pth")
        return jsonify({"success": True})
    return jsonify({"success": False})



@app.route("/image-analysis", methods = ["POST"])
def analyze_image():
    if not check_auth(): abort(401, "Error! Not
authorized!")

    imageBase64 = request.json.get('image')
    print(imageBase64)
    with open("temp.png", "wb") as f:
        f.write(base64.b64decode(imageBase64))
```

```
        res = detector.check_herpes("temp.png")
        return jsonify({"result": res})


if __name__ == '__main__':
    app.run(port=3001, debug=True)
```

Листинг А.2 – Файл «DockerFlask.py»

```
FROM python:3.11

WORKDIR /usr/src/app

COPY ./requirements.txt /usr/src/app/requirements.txt

RUN pip install -r requirements.txt

COPY . .

ENTRYPOINT ["python"]

CMD ["app.py"]
```

Листинг А.3 – Файл «DockerComposeFlask.py»

```
version: "3.4"

services:
  medical-app:
    image: python:3.11
    build:
      context: .
    volumes:
      - ./storage:/usr/src/app/storage
    ports:
```

```
      - "3001:3001"
    restart: always
    networks:
      - my-network

networks:
  my-network:
    driver: bridge
```

Листинг А.4 – Файл «Optimizer.ipynb»

```
%pip install torch torchvision torchsummary numpy matplotlib
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import torch.backends.cudnn as cudnn
import numpy as np
import torchvision
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
import time
import os
from PIL import Image
from tempfile import TemporaryDirectory

cudnn.benchmark = True
plt.ion()   # interactive mode

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

```
    elif torch.backends.mps.is_built() and
torch.backends.mps.is_available():
        device = torch.device("mps")
    print(device)


    import zipfile
    with zipfile.ZipFile('/content/drive/MyDrive/archive.zip',
'r') as zip_ref:
        zip_ref.extractall('/content/drive/MyDrive')


    # Data augmentation and normalization for training
    # Just normalization for validation
    data_transforms = {
        'train': transforms.Compose([
            transforms.RandomResizedCrop(224),
            transforms.RandomHorizontalFlip(),
            transforms.ToTensor(),
            transforms.Normalize([0.485, 0.456, 0.406], [0.229,
0.224, 0.225])
        ]),
        'test': transforms.Compose([
            transforms.Resize(256),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
            transforms.Normalize([0.485, 0.456, 0.406], [0.229,
0.224, 0.225])
        ]),
    }


    data_dir = 'data/'
    image_datasets = {x:
datasets.ImageFolder(os.path.join(data_dir, x),
```

```
data_transforms[x])
                      for x in ['train', 'test']}
    dataloaders = {x:
torch.utils.data.DataLoader(image_datasets[x], batch_size=4,
                                                 shuffle=True,
num_workers=4)
                  for x in ['train', 'test']}
    dataset_sizes = {x: len(image_datasets[x]) for x in
['train', 'test']}
    class_names = image_datasets['train'].classes


    device = torch.device("cuda:0" if torch.cuda.is_available()
else "cpu")


    def imshow(inp, title=None):
        """Display image for Tensor."""
        inp = inp.numpy().transpose((1, 2, 0))
        mean = np.array([0.485, 0.456, 0.406])
        std = np.array([0.229, 0.224, 0.225])
        inp = std * inp + mean
        inp = np.clip(inp, 0, 1)
        plt.imshow(inp)
        if title is not None:
            plt.title(title)
        plt.pause(0.001)  # pause a bit so that plots are
updated



    # Get a batch of training data
    inputs, classes = next(iter(dataloaders['train']))


    # Make a grid from batch
```

```
    out = torchvision.utils.make_grid(inputs)


    imshow(out, title=[class_names[x] for x in classes])


    def train_model(model, criterion, optimizer, scheduler,
num_epochs=25):
        since = time.time()


        # Create a temporary directory to save training
checkpoints
        with TemporaryDirectory() as tempdir:
            best_model_params_path = os.path.join(tempdir,
'best_model_params.pt')


            torch.save(model.state_dict(),
best_model_params_path)
            best_acc = 0.0


            for epoch in range(num_epochs):
                print(f'Epoch {epoch}/{num_epochs - 1}')
                print('-' * 10)


                # Each epoch has a training and validation phase
                for phase in ['train', 'test']:
                    if phase == 'train':
                        model.train()  # Set model to training
mode
                    else:
                        model.eval()   # Set model to evaluate
mode


                    running_loss = 0.0
                    running_corrects = 0
```

```
                    # Iterate over data.
                    for inputs, labels in dataloaders[phase]:
                        inputs = inputs.to(device)
                        labels = labels.to(device)

                        # zero the parameter gradients
                        optimizer.zero_grad()

                        # forward
                        # track history if only in train
                        with torch.set_grad_enabled(phase ==
'train'):
                            outputs = model(inputs)
                            _, preds = torch.max(outputs, 1)
                            loss = criterion(outputs, labels)

                            # backward + optimize only if in
training phase
                            if phase == 'train':
                                loss.backward()
                                optimizer.step()

                        # statistics
                        running_loss += loss.item() *
inputs.size(0)
                        running_corrects += torch.sum(preds ==
labels.data)
                    if phase == 'train':
                        scheduler.step()

                    epoch_loss = running_loss /
dataset_sizes[phase]
```

```
                    epoch_acc = running_corrects.double() /
dataset_sizes[phase]

                    print(f'{phase} Loss: {epoch_loss:.4f} Acc:
{epoch_acc:.4f}')

                    # deep copy the model
                    if phase == 'test' and epoch_acc > best_acc:
                        best_acc = epoch_acc
                        torch.save(model.state_dict(),
best_model_params_path)

                print()

            time_elapsed = time.time() - since
            print(f'Training complete in {time_elapsed //
60:.0f}m {time_elapsed % 60:.0f}s')
            print(f'Best val Acc: {best_acc:4f}')

            # load best model weights

model.load_state_dict(torch.load(best_model_params_path))
        return model


    def visualize_model(model, num_images=6):
        was_training = model.training
        model.eval()
        images_so_far = 0
        fig = plt.figure()

        with torch.no_grad():
            for i, (inputs, labels) in
enumerate(dataloaders['test']):
```

```
                inputs = inputs.to(device)
                labels = labels.to(device)

                outputs = model(inputs)
                _, preds = torch.max(outputs, 1)

                for j in range(inputs.size()[0]):
                    images_so_far += 1
                    ax = plt.subplot(num_images//2, 2,
images_so_far)
                    ax.axis('off')
                    ax.set_title(f'predicted:
{class_names[preds[j]]}')
                    imshow(inputs.cpu().data[j])

                    if images_so_far == num_images:
                        model.train(mode=was_training)
                        return
            model.train(mode=was_training)

    model_ft = models.resnet50(weights='IMAGENET1K_V2')
    num_ftrs = model_ft.fc.in_features
    # Here the size of each output sample is set to 2.
    # Alternatively, it can be generalized to
``nn.Linear(num_ftrs, len(class_names))``.
    model_ft.fc = nn.Linear(num_ftrs, 23)

    model_ft = model_ft.to(device)

    criterion = nn.CrossEntropyLoss()

    # Observe that all parameters are being optimized
```

```
    optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.005,
momentum=0.9)


    # Decay LR by a factor of 0.1 every 7 epochs

    exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft,
step_size=7, gamma=0.1)


    model_ft = train_model(model_ft, criterion, optimizer_ft,
exp_lr_scheduler,
                           num_epochs=50)


    visualize_model(model_ft)
```

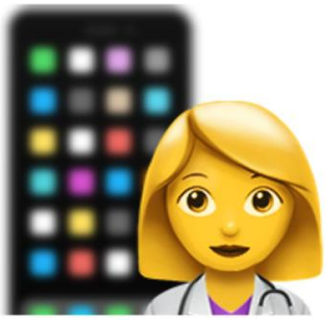
ПРИЛОЖЕНИЕ Б

(рекомендуемое)

Медицинское приложение. Модуль первичной диагностики заболевания

Презентация

Медицинское приложение. Модуль первичной диагностики



Выполнила: Максимова Д.Р., гр. 22ВВИм1

Руководитель: Митрохин М.А.

Цель магистерской диссертации

Разработка модуля первичной диагностики заболеваний по изображениям с применением технологий искусственного интеллекта. Исследование архитектур и моделей нейронных сетей для применения в сфере медицины

Диаграмма последовательности

Структурная диаграмма

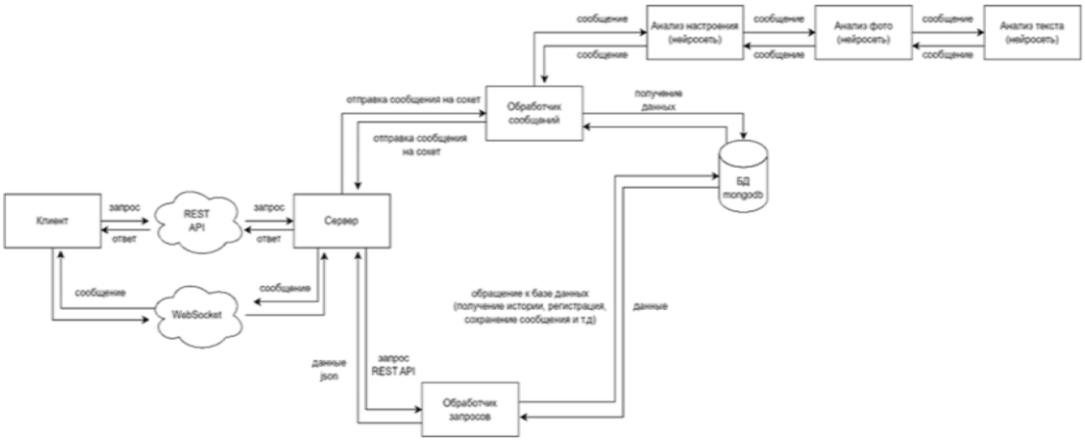
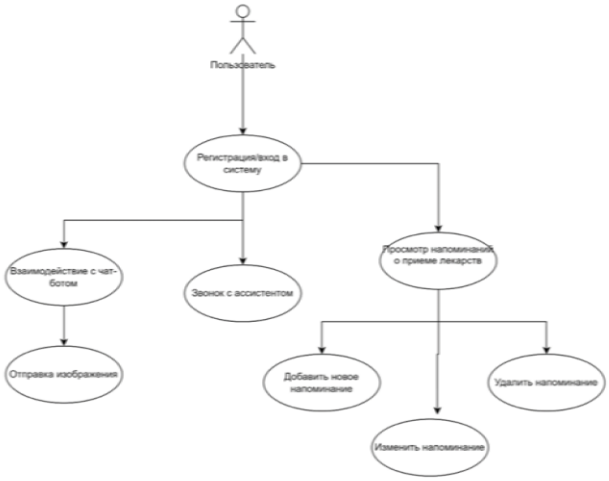


Диаграмма вариантов использования



Стек технологий

PyTorch

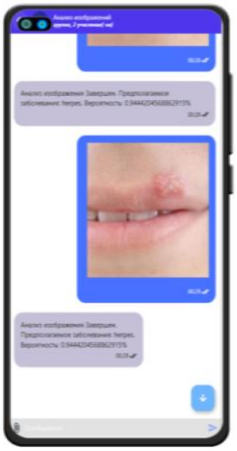
Docker

Python

Git

Flutter

Первичная диагностика по фото

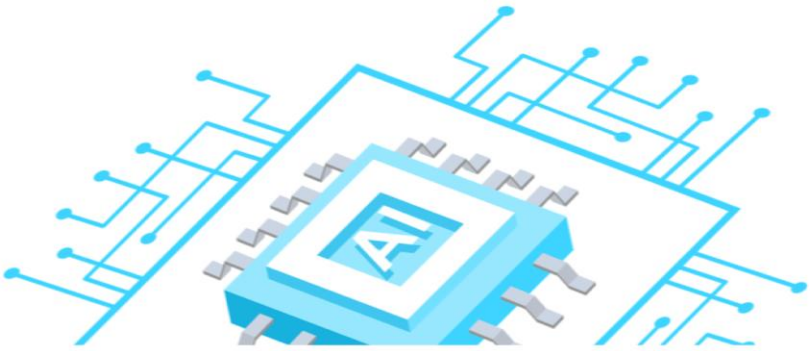


Для определения заболевания по изображениям, пользователю необходимо нажать на кнопку «Скрепка» в левом нижнем углу экрана. После нажатия на кнопку откроется галерея телефона. Пользователю необходимо выбрать необходимо фото. После выбора фотография автоматически отправится в чат-бот. Чат-бот в ответ дает информацию о предполагаемом заболевании

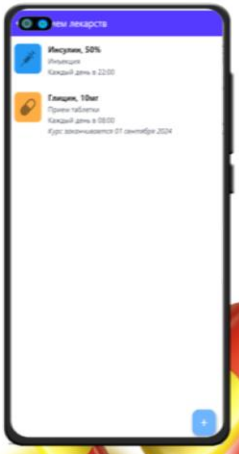
Архитектура нейронной сети: Faster R-CNN
Модель: Resnet50
Точность: 70%

Сверточные нейронные сети

R-CNN – двухступенчатый детектор, который упрощает определение объектов на изображениях. Карта объектов вычисляется только один раз для всего изображения. Region of Interests (RoI) – этот слой определяет карты объектов стандартного размера для возможных областей. Затем выполняется классификация и точное определение области, где находится объект

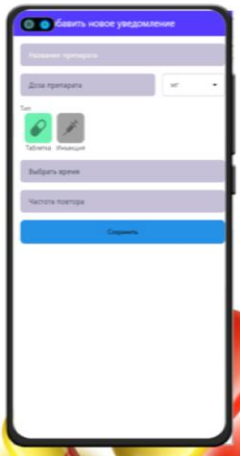


Напоминание о приеме лекарств



На странице «Прием лекарств» отображаются те лекарства, для которых пользователь установил напоминания. Чтобы добавить напоминание о приеме лекарства необходимо нажать на кнопку «+» в правом нижнем углу экрана. Если у пользователя добавлены напоминания о приеме лекарств, тогда к нему на телефон приходят уведомления по принципу будильника

Напоминание о приеме лекарств



На странице «Добавить уведомление» пользователь вводит все необходимые данные для добавление напоминания о приеме лекарства

Звонок с ассистентом

При нажатии на кнопку «Звонок» система начинает анализировать речь пользователя и передавать данные для нейронной сети для генерации диалога с пользователем. В ответ на вопрос пользователю с сервера приходит текст, который переводится в голосовое сообщение на клиенте

Заключение

Разработанная подсистема является частью общей комплексной системы, которая необходима для диагностики пользователя, путем общения с чат-ботом, который генерирует текст

Разработанная подсистема может помочь пользователю следить за здоровьем, пользователь может настраивать уведомления для приема таблеток или применения инъекций и может существенно помочь человеку своевременно и быстро проверить собственный организм для диагностики

В будущем планируется расширить функционал подсистемы, улучшить точность нейронной сети, увеличить количество классов заболеваний, которые будут определяться