МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра информационных систем управления

КОСТЕЦКИЙ Павел Сергеевич

РАЗРАБОТКА ПРИЛОЖЕНИЯ ДИСТАНЦИОННОЙ ДИАГНОСТИКИ ЗЛОКАЧЕСТВЕННЫХ НОВООБРАЗОВАНИЙ

Дипломная работа

Научный руководитель:
Старший преподаватель
С.Е. Гутников

Допущена к защите
«___»_____20___г.
Заведующий кафедрой информационных систем управления
Доктор технических наук, доцент А.М. Недзьведь

Минск, 2025

ОГЛАВЛЕНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ И ТЕРМИНОВ

ВОЗ – всемирная организация здравоохранения

ИНС – искусственная нейронная сеть

Adam – алгоритм оптимизации ИНС

Android – мобильная операционная система

API – интерфейс для взаимодействия между сервером и клиентом (Application Programming Interface)

COVID-19 – коронавирусная инфекция (упоминается в контексте развития телемедицины)

DermNet – DermNet NZ (онлайн-ресурс по дерматологии)

DICOM – стандарт для медицинских изображений (Digital Imaging and Communications in Medicine)

EfficientNet – семейство архитектур нейронных сетей

ER-диаграмма – диаграмма сущность-связь для проектирования БД (Entity-Relationship)

F1-Score – гармоническое среднее точности и полноты

Flask – веб-фреймворк на Python для серверной части

GUI – графический интерфейс пользователя (Graphical user interface)

ISIC – международная база данных кожных изображений International Skin Imaging Collaboration)

HTTP – протокол передачи данных (HyperText Transfer Protocol)

JSON – формат обмена данными (JavaScript Object Notation)

JWT – токен для аутентификации пользователей (JSON Web Token)

Kaggle – платформа для соревнований и обмена данными в области анализа данных

Keras – высокоуровневый API для работы с нейросетями

Kotlin – язык для разработки Android-клиента

NASA – национальное управление по аэронавтике и исследованию космического пространства (National Aeronautics and Space Administration)

OkHttp – Библиотека для HTTP-запросов в Android

PostgreSQL – реляционная база данных

ReLU – функция активации (Rectified Linear Unit)

ResNet – архитектура нейронных сетей (Residual Network)

REST – архитектурный стиль веб-сервисов (Representational State Transfer)

Retrofit – библиотека для сетевых запросов в Android

Roboflow Universe – платформа для публикации и использования наборов данных для компьютерного

ROC-AUC – кривая ошибок и площадь под ней

TensorFlow – библиотека для машинного обучения

YOLO – архитектура нейронных сетей для обнаружения объектов (You Only Look Once)

РЕФЕРАТ

Структура и объём дипломной работы

68 страниц, 11 рисунков, 4 таблицы, 9 приложений, 14 источников

Ключевые слова**:** TENSORFLOW, KERAS, PYTHON, KOTLIN, REST, ИСКУССТВЕННАЯ НЕЙРОННАЯ СЕТЬ, МЕЛАНОМА, КОЖНЫЕ НОВООБРАЗОВАНИЯ, МЕДИЦИНА, ТЕЛЕМЕДИЦИНА, ДИСТАНЦИОННАЯ ДИАГНОСТИКА.

Объект исследования – телемедицина, использование искусственной нейронной сети для предварительной диагностики.

Предмет исследования – разработка системы дистанционной диагностики злокачественных новообразований.

Цель исследования – обозначить проблемы современной медицины в аспекте доступности, разработать систему дистанционной диагностики злокачественных новообразований с использованием искусственной нейронной сети.

Методы исследования – системный подход, изучение соответствующей литературы в области телемедицины, компьютерного зрения и глубокого обучения, программная реализация на языке программирования Kotlin, Python с использованием библиотек TensorFlow и Ultralytics.

Полученные результаты – разработанная целевая система по дистанционной диагностике кожных новообразований с помощью искусственной нейронной сети и дистанционной консультации с врачом.

Достоверность материалов и результатов дипломной работы – использованные материалы и результаты дипломной работы являются достоверными. Работа выполнена самостоятельно.

Область возможного практического применения – разработанная система может быть использована для систем по дистанционному диагностированию кожных новообразований.

РЭФЕРАТ

Структура і аб’ём дыпломнай працы

68 старонак, 11 малюнкаў, 4 табліцы, 9 дадаткаў, 14 крыніц

Ключавыя словы: TENSORFLOW, KERAS, PYTHON, KOTLIN, REST, ШТУЧНАЯ НЕЙРОННАЯ СЕТКА, МЕЛАНОМА, СКУРНЫЯ НАВАЎТАВАННЯ, МЕДЫЦЫНА, ТЭЛЕМЕДЫЦЫНА, ДЫСТАНЦЫЙНАЯ ДЫЯГНАСЦЬ.

Аб'ект даследавання – тэлемедыцына, выкарыстанне штучнай нейронавай сеткі для папярэдняй дыягностыкі.

Прадмет даследавання – распрацоўка сістэмы дыстанцыйнай дыягностыкі злаякасных новаўтварэнняў.

Мэта даследавання – пазначыць праблемы сучаснай медыцыны ў аспекце даступнасці, распрацаваць сістэму дыстанцыйнай дыягностыкі злаякасных новаўтварэнняў з выкарыстаннем штучнай нейронанай сеткі.

Метады даследавання – сістэмны падыход, вывучэнне адпаведнай літаратуры па галіне тэлемедыцыны, камп'ютарнага зроку і глыбокага навучання, праграмная рэалізацыя на мове праграмавання Kotlin, Python з выкарыстаннем бібліятэк TensorFlow і Ultralytics.

Атрыманыя вынікі – распрацаваная мэтавая сістэма па дыстанцыйнай дыягностыкі скурных наватвораў з дапамогай штучнай нейронавай сеткі і дыстанцыйнай кансультацыі з лекарам.

Дакладнасць матэрыялаў і вынікаў дыпломнай работы – выкарыстаныя матэрыялы і вынікі дыпломнай работы з'яўляюцца дакладнымі. Праца выканана самастойна.

Вобласць магчымага практычнага прымянення – распрацаваная сістэма можа быць выкарыстана для сістэм па дыстанцыйным дыягнаставанні скурных наватвораў.

SUMMARY

Structure and scope of the diploma work

68 pages, 11 figures, 4 tables, 9 appendixes, 14 references

Keywords: TENSORFLOW, KERAS, PYTHON, KOTLIN, REST, ARTIFICIAL NEURAL NETWORK, MELANOMA, SKIN NEOPLASMS, MEDICINE, TELEMEDICINE, REMOTE DIAGNOSTICS.

Object of the research – telemedicine, the use of artificial neural network for preliminary diagnostics.

Subject of the research – development of a system for remote diagnostics of malignant neoplasms.

The aim of the research – is to identify the problems of modern medicine in terms of accessibility, to develop a system for remote diagnostics of malignant neoplasms using artificial neural network.

Research methods – a systematic approach, a study of relevant literature in the field of telemedicine, computer vision and deep learning, software implementation in the Kotlin programming language, Python using the TensorFlow and Ultralytics libraries.

The obtained results – are a developed target system for remote diagnostics of skin neoplasms using artificial neural networks and remote consultation with a doctor.

Authenticity of the materials and results of the diploma work – the materials used and the results of the diploma work are authentic. The work has been put through independently.

Recommendation on the usage – the developed system can be used for systems for remote diagnostics of skin neoplasms.

ВВЕДЕНИЕ

Ускоренное развитие цифровых технологий напрямую влияет на скорость развития медицины. Главным образом это влияет на такие факторы, как качество медицины и своевременность выявления заболеваний с их последующим лечением. В частности, одним из важнейших направлений в медицине можно считать онкологию кожи. Правильная и своевременная диагностика кожных заболеваний позволяет повысить вероятность успешного лечения до невероятных значений.

Актуальность данной темы обуславливается достаточно высокими показателями распространенности и смертности от кожных заболеваний. По данным Всемирной организации здравоохранения в 2020 году по всему миру было зарегистрировано более чем полтора миллиона случаев рака кожи и более ста двадцати тысяч обусловленных ими случаем смерти. В частности, из-за чрезмерного воздействия ультрафиолетового излучения выявлено примерно 300 тысяч случаев меланомы кожи, включая почти 60 тысяч случаев с летальным исходом. Что составляет показатель смертности в 20%. [2]

Рак кожи является одним из самых распространенных типов онкологических заболеваний. Если признаки данных заболеваний обнаружены рано, на стадии, когда болезнь ещё не успела развиться, то процесс лечения существенно упрощается. Множество врачей и специалистов в области обследований здравоохранения рассматривают осмотр кожи в качестве части плановых медицинских обследований. [4] Вследствие чего поднимается вопрос развития телемедицины: доступности оказания медицинских услуг, экономии времени для своевременного лечения, снижения нагрузки на медицинских персонал и возможности проводить удаленные консультации.

Целью данной работы является разработка системы дистанционной диагностики злокачественных новообразований. Данную систему в дальнейшем можно будет использовать для дистанционной диагностики различных кожных новообразований, удаленных консультаций и предварительных диагностик посредством искусственных нейронных сетей.

Выполнение данной работы будет происходить посредством исследования и анализа научной литературы, использования специализированных библиотек и инструментария, необходимого для реализации целевой системы.

ГЛАВА 1
ПОСТАНОВКА ПРИКЛАДНОЙ ЗАДАЧИ

1.1 Постановка задач

В качестве основной цели данной работы выступает разработка приложения, которое будет состоять из трёх компонент: серверной, клиентской для пациентов и клиентской для врачей. Поставленные задачи требуют реализовать возможность передачи изображений на сервер с последующим использованием мощностей серверной машины и ИНС. Впоследствии пациент будет иметь возможность получать первичную диагностику от ИНС, а также иметь возможность участвовать в дистанционных консультациях.

Окончательная версия будущего приложения будет являться упрощенной в направлении своей специализации, а именно диагностике злокачественных новообразований кожи. Однако приложение будет спроектировано таким образом, чтобы впоследствии оно могло получить развитие с помощью легкого масштабирования.

В результате достижения данной цели мы должны получить:

- базу данных, в которой будут храниться фото, результаты первичной диагностики, данные о пациентах, врачах и история сообщений дистанционной консультации;
- обученную ИНС, которая будет способна автоматически классифицировать медицинские изображения кожных новообразований, разделяя их на две категории: злокачественные(меланома) и доброкачественные образования. Полученный компонент, содержащий ИНС, будет интегрирован в целевую систему дистанционной диагностики кожных новообразований, обеспечив точность, удобство, скорость и доступность;
- серверную часть, которая будет содержать ИНС, взаимодействовать с базой данных, заниматься передачей данных между двумя клиентами будущей системы: клиентом для врачей и клиентом для пациентов;
- клиентскую часть для пациентов, в которой можно будет заходить в свой профиль и получать первичную диагностику кожного новообразования от ИНС по фотографии и обмениваться сообщениями с врачом дистанционно;
- клиентская часть для врачей, в которой врачи также могут заходить в свой профиль, просматривать список своих пациентов и отдельно каждому пациенту присылать сообщения об их дистанционных заключениях и дальнейших рекомендациях.

Чтобы решить данную задачу необходимо проделать следующий перечень действий:

- изучить современные подходы, стили и методы, применяемые для проектирования клиент-серверного программного обеспечения;
- выбрать стек технологий и инструментов для достижения поставленной цели;
- обучить искусственную нейронную сеть;
- разработать архитектуру базы данных;
- разработать серверную часть;
- интегрировать ИНС в серверную часть;
- разработать клиентскую часть для пациентов;
- разработать клиентскую часть для врачей;
- решить первичный вопрос безопасности данных.

1.2 Требования к системе

Разрабатываемая система представляет собой программный комплекс для автоматизированной первичной диагностики злокачественных новообразований кожи на основе искусственных нейронный сетей. Система состоит из серверной части с интегрированной ИНС, клиентского приложения для пациентов, клиентского приложения для врачей и базы данных для хранения медицинской информации. Основная цель разработки – создание удобного, безопасного и точного инструмента для раннего выявления меланомы.

Серверный компонент должен обеспечивать:

- прием и валидацию цифровых изображений кожных покровов;
- обработку изображений с использованием заранее обученной ИНС;
- классификацию новообразований с определением степени точности поставленного диагноза;
- взаимодействие с клиентскими компонентами системы;
- возможность коммуникации двух клиентов между собой, где один из них – это клиент врача, а другой – пациента.

Клиентская компонента системы пациента должен иметь:

- систему аутентификации и авторизации пользователей;
- функционал для съемки и загрузки изображений;
- отображение результатов первичной диагностики;
- коммуникацию с врачом;

Клиентский компонент системы врача должен иметь:

- систему аутентификации и авторизации пользователей;

- функционал загрузки изображений, их отображения вместе с результатами первичной диагностики посредством ИНС;

- просмотр списка пациентов;

- коммуникацию с пациентами.

Требования к системе хранения данных:

- реляционная структура для хранения пользовательских данных и результатов диагностики.

1.3 Выводы

В данной главе поставлены цели и задачи текущей работы. Были сформулированы требования к разрабатываемой системе.

Главной задачей становится проектирование базы данных, разработка клиентов для пациентов и врачей, а также сервера. Необходимо также спроектировать и обучить ИНС для первичной диагностики и её интеграции в серверную часть. В целевой системе также будет реализована возможность взаимодействия врачей и пациентов для осуществления дистанционных консультаций.

Реализация данных задач позволит создать основу для дальнейшей разработки комплексной системы дистанционной диагностики кожных заболеваний, что имеет важное значение для повышения качества медицинского обслуживания и доступности диагностики. Разрабатываемая система представляет собой специализированное решение для первичной диагностики кожных новообразований.

ГЛАВА 2
ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ТЕЛЕМЕДИЦИНЫ

2.1 Определение, сущность и классификация телемедицины

Телемедицина представляет собой стремительно развивающуюся область здравоохранения, которая использует современные информационно-коммуникационные технологии для преодоления географических, временных и организационных барьеров в оказании медицинской помощи. Согласно ВОЗ, телемедицина – это предоставление услуг здравоохранения в условиях, когда расстояние является критическим фактором, с использованием информационно-коммуникационных технологий для обмена информацией в диагностических, лечебных и профилактических целях. [14]

Следующие аспекты можно считать ключевыми в определении сущности телемедицины:

- дистанционный характер, то есть оказание медицинских услуг без присутствия пациента и врача в одном физическом месте;
- использование цифровых технологий: применение специальных программ, телекоммуникационного оборудования и медицинских устройств;
- обмен информацией в виде различных цифровых форматов (текст, цифровые изображения, видео, сигналы);
- клиническая направленность – решение практических медицинских задач диагностики, лечения и профилактики.

В вопросе классификации телемедицинских технологий существует несколько подходов. Самый популярный вариант классификации основан на характере взаимодействия сторон:

- синхронная телемедицина включает в себя телефонные консультации, онлайн-чаты, видеоконференцсвязь в реальном времени. Отличительной чертой является требование участия одного врача и одного пациента;
- асинхронная телемедицина включает в себя передачу медицинских данных с последующей их аналитикой, дистанционную интерпретацию диагностических исследований, электронные консультации по электронной почте или через специализированные платформы. Отличительная черта здесь – это отсутствие требования одновременного присутствия врача и пациента;
- дистанционный мониторинг включает в себя сбор и передачу физиологических параметров пациента, использование носимых устройств и датчиков, применение мобильных медицинских приложений. Отличительная черта здесь – это непрерывный или периодический контроль состояния пациента.

Другой подход в вопросе классификации состоит в классификации по направлению информационных потоков. Бывают следующие виды:

- врач-пациент – прямое общение специалиста с пациентом;
- врач-врач – консультации между медицинскими специалистами;
- система-пациент – автоматизированные системы мониторинга и диагностики состояния;
- система-врач – автоматизированные системы принятия решений в области медицины.

2.2 Исторические этапы становления и развития телемедицины

2.2.1 Период первых экспериментов (1950-1970-е годы)

Зарождение телемедицины началось в середине прошлого века, когда стали возможными передача информации через телекоммуникационные технологии. Первые попытки передачи медицинской информации на расстоянии были сделаны в 1950-е годы в США и Канаде, где экспериментировали с передачей радиосигналов, которые содержали данные о здоровье человека. [9] Уже в этот период в медицине произошло несколько значимых событий:

- 1959 г. – в университете Небраски появилась первая в мире двухсторонняя видеоконференцсвязь для психиатрических консультаций;
- 1964 г. – NASA разработало систему биотелеметрии для мониторинга состояния организма космонавтов;
- 1967 г. – Массачусетская больница по телефонной линии передала первые оцифрованные рентгенограммы.

Для того, чтобы передать данные на большие расстояния, врачи использовали аналоговые технологии, что неудивительною. Они попросту не могли больше – по аналогу передаются сигналы низкого, специфичного разрешения.

2.2.2 Становление цифровой телемедицины (1980-1990-е годы)

Дальнейшее развитие телемедицины (1980-1990-е годы) завершило этот этап. Весь мир перешел на цифровые технологии, открыв впереди много возможностей. Важные достижения:

- разработка стандартов цифровой передачи медицинских изображений (DICOM);

- создание первых специализированных телемедицинских сетей;

- первое появление коммерческих телемедицинских систем.

В 1990-е годы, с развитием интернета, телемедицина получила новый импульс развития. В 1993 году произошел запуск первой в мире телемедицинской интернет-системы для консультации. В 1995 году состоялся запуск поддержки космонавтов программой “Телемедицина в космосе”, созданной NASA. А в 1997 году прошла первая успешная телехирургическая операция (проект “Lindbergh”).

2.2.3 Современный этап (2000-е годы – настоящее время)

Современное стремительное развитие телемедицины начинается после 2000-х годов, когда внедряется мобильная связь и облачные технологии, что дает телемедицине совершенно иную перспективу. Это подтверждают следующие события:

- демократизация доступа: появление мобильных приложений для здоровья, развитие носимых медицинских устройств, создание платформ для масштабных теле-консультаций;

- интеграция искусственного интеллекта: алгоритмы анализа медицинских изображений, системы поддержки принятия решения, чат-боты как первый врач;

- глобализация услуг телемедицины: международные телемедицинские сети, кросс-граничные консультации, стандартизация протоколов обмена данными.

Особый импульс развитию телемедицины придала пандемия COVID-19, когда потребность в дистанционных медицинских услугах возросла в разы. По данным исследований в 2020 году количество теле-консультаций увеличилось на 300-400% по сравнению с предыдущими годами. [10]

2.3 Преимущества и перспективы развития

2.3.1 Преимущества для пациентов

Телемедицинские технологии устраняют географические барьеры, обеспечивая равный доступ к квалифицированной медицинской помощи для жителей отдалённых и труднодоступных регионов. Это особенно значимо для

пациентов, проживающих в сельской местности и территориях с низкой плотностью медицинских учреждений. [6]

Применение дистанционных консультационных сервисов позволяет существенно сократить транспортные издержки и временные потери, связанные с посещением медицинских организаций. Это особенно актуально для пациентов с ограниченной мобильностью и лиц пожилого возраста.

Телемедицинские платформы обеспечивают возможность экстренного получения врачебных консультаций в режиме реального времени, что критически важно при острых состояниях и неотложных ситуациях.

2.3.2 Преимущества для медицинских работников

Телемедицинские технологии создают условия для постоянного профессионального роста через участие в дистанционных образовательных программах, вебинарах и онлайн-конференциях.

Цифровые платформы обеспечивают возможность оперативного консультирования сложных случаев с ведущими специалистами, способствуя принятию более обоснованных клинических решений.

Внедрение телемедицинских сервисов позволяет рационализировать нагрузку на медицинский персонал, сократить время ожидания приёма и повысить эффективность использования рабочего времени.

Возможность удаленного мониторинга состояния пациентов способствует повышению приверженности лечению и позволяет своевременно выявлять негативные тенденции в течении заболеваний.

2.3.3 Преимущества для системы здравоохранения

Телемедицинские технологии способствуют более эффективному распределению кадровых, материальных и финансовых ресурсов здравоохранения.

Стандартизация диагностических и лечебных процессов через телемедицинские платформы приводит к улучшению показателей эффективности лечения и удовлетворённости пациентов. [5]

Дистанционные технологии обеспечивают преодоление территориального неравенства в доступности медицинской помощи, что особенно значимо для регионов с дефицитом специалистов. [13]

Телемедицина создаёт условия для концентрации экспертных знаний и дорогостоящего оборудования в референс-центрах с возможностью их удалённого использования. Это позволяет обеспечить высокий стандарт медицинской помощи на всей территории обслуживания.

2.4 Выводы

В данной главе были рассмотрены теоретические основы телемедицины как современного направления развития здравоохранения. В ходе исследования были определены ключевые аспекты понятия телемедицины, включая её дистанционный характер, использование цифровых технологий, информационный обмен и клиническую направленность. Проведена детальная классификация телемедицинских технологий по характеру взаимодействия участников и направлению информационных потоков.

Был осуществлен комплексный анализ исторических этапов становления телемедицины.

В главе систематизированы преимущества телемедицины для различных участников системы здравоохранения. Выявлены ключевые выгоды, включая повышение доступности медицинской помощи, оптимизацию временных и финансовых затрат, возможности профессионального развития медицинских специалистов и рационализацию использования ресурсов системы здравоохранения.

ГЛАВА 3
ОБЗОР СПЕЦИАЛИЗИРОВАННЫХ БИБЛИОТЕК И ИНСТРУМЕНТОВ

В разработке системы были использованы два ключевых языка программирования Python и Kotlin. В качестве базы данных было решено использовать PostgreSQL.

Язык программирования Python выбран в качестве основного языка для серверной части системы и клиентской части для врачей. Он был выбран благодаря следующим характеристикам:

- богатая экосистема библиотек для веб-разработки и машинного обучения;
- кроссплатформенность.

Основные задачи, решаемые Python-компонентом:

- обработка бизнес-логики приложения;
- взаимодействие с базой данных;
- реализация REST API для мобильного клиента и будущего клиента для врачей;
- выполнение ресурсоемких вычислений;
- управление аутентификацией и авторизацией.

Язык программирования Kotlin был выбран для разработки мобильного клиента на операционной системе Android благодаря:

- безопасности типов и null-безопасности;
- современным языковым конструкциям;
- полной интероперабельности с Java.

Основные задачи Kotlin-компонента:

- реализация пользовательского интерфейса;
- взаимодействие с серверными API;
- работа с аппаратными возможностями устройства.

3.1 Библиотеки серверной части и клиента для врачей

Серверная часть приложения, как уже было упомянуто, разработана на языке программирования Python. Далее вкратце описаны основные библиотеки, которые использовались при разработке сервера.

3.1.1 Веб-фреймворк Flask [7]

Библиотека Flask использовалась для создания RESTful API сервера. Ключевые особенности данной библиотеки, необходимые при разработке сервера, включают в себя:

- обработку входящих от клиентов http-запросов;
- преобразования данных в JSON формат для последующей их передачи между клиентом и сервером;
- работу с файлами, которая будет необходима в дальнейшем для отправки на клиент врача фотографий.

3.1.2 Аутентификация JWT

Для реализации механизма аутентификации использовалось расширение Flask-JWT-Extended. Оно позволит реализовать авторизацию и аутентификацию в REST API. Это расширение позволяет управлять, генерировать и верифицировать токены. А также получать данные пользователей из этих самых токенов. [8]

3.1.3 Работа с базой данных

Для взаимодействия с PostgreSQL использовалась библиотека psycopg2. Основные операции для работы с базой данных включают в себя установку соединения, выполнение SQL-запросов и поддержку принципа транзакции.

3.1.4 Библиотека Tkinter

Библиотека Tkinter в Python используется для создания GUI приложений. Данная библиотека позволяет создавать различные части графического интерфейса, такие как кнопки, текстовые поля, выпадающие меню и т.д. В общем Tkinter является простым и эффективным инструментом для создания графических интерфейсов в Python. Именно с помощью этой библиотеки реализован клиент для врачей в виде десктопного приложения с интерфейсом, создание которого обеспечивает данная библиотека.

3.2 Машинное обучение

Для интеграции ИНС в целевую систему сначала необходимо спроектировать модель ИНС, обучить её и затем использовать для конечной цели. Весь процесс происходит отдельно и для него нужны свои дополнительные библиотеки и инструментарий.

3.2.1 Среда разработки Google Colab

Google Colab – это размещённый удаленно сервис Jupyter Notebool, не требующий настройки для использования и предоставляющий бесплатный доступ к вычислительным ресурсам. Colab особенно хорошо подходит для задач машинного обучения, что и требуется. Именно поэтому данная облачная среда и была выбрана для обучения искусственной нейронной сети.

3.2.2 Библиотека TensorFlow

TensorFlow – это среда машинного обучения с открытым исходным кодом, которая предоставляет возможность создания и обучения моделей ИНС. Она используется для внедрения машинного обучения и глубокого обучения.

Интерфейс, который предлагает TensorFlow и который необходим для решения поставленной задачи, — это Keras. Keras – это высокоуровневый API, подходящий для быстрого прототипирования. Он упрощает разработку моделей ИНС и предоставляет доступ к множеству готовых архитектур ИНС.

Также стоит упомянуть, что TensorFlow предоставляет инструментарий для предобработки данных, включая процессы аугментации и нормализации.

3.2.3 Библиотека Ultralytics

Библиотека Ultralytics предоставляет доступ к последним версиям ИНС YOLO, включая YOLOv8 и YOLOv11. Данные нейронные сети также будут проверятся на предмет эффективности решения поставленной задачи, поскольку в них реализована возможность решения задач классификации.

3.2.4 Библиотека Scikit-learn

Для оценки эффективности исследуемых моделей ИНС будет использоваться данная библиотека, которая предлагает полный набор инструментов для анализа качества классификации. С её помощью можно оценить желаемые метрики, что позволяет определить качество работы той или иной модели.

3.3 Библиотеки и инструменты мобильного клиента

Мобильное приложение разработано на Kotlin с использованием библиотек OkHttp и Retrofit. Их необходимость заключается в выполнении http-запросов и соответствии архитектуре REST. Также необходимо использовать различные методы для работы с JSON форматом.

3.4 Выводы

В данной главе были рассмотрены главные используемые библиотеки и языки программирования, предоставляющие необходимый инструментарий для разработки системы дистанционного диагностирования. На языке программирования Python будет создана серверная часть и клиент для врачей, а также обучена ИНС. На языке программирования Kotlin будет разработан клиент для пациентов, работающий на операционной системе Android, что сделает его легкодоступным.

ГЛАВА 4
ОБУЧЕНИЕ ИСКУССТВЕННОЙ НЕЙРОННОЙ СЕТИ

4.1 Сбор и подготовка исходных данных

4.1.1 Обзор источников данных

Для обучения любой ИНС в самом начале необходимо получить исходные данные, на которых и будет происходить обучение. Поскольку ранее упоминалось, что конечная система в своей минимальной комплектации должна уметь выполнять предварительную диагностику на определение меланомы по цифровым изображениям, то исходные данные будем искать в соответствующих открытых источниках. В них нам необходимо найти датасеты с двумя классами изображений: злокачественные и доброкачественные кожные новообразования.

Найти необходимые нам данные можно на следующих интернет ресурсах:

ISIC - это крупный и постоянно расширяющийся открытый архив изображений кожи, который служит общественным ресурсом для обучения, исследований и разработки диагностических алгоритмов искусственного интеллекта. В качестве преимуществ данного источника данных можно выделить его большой объем, доступность и разнообразие данных. Ключевой недостаток данного источника данных – это предоставляемое качество изображений. Многие изображения содержат артефакты или же просто сами являются изображениями низкого качества.

DermNet — это ведущий ресурс по дерматологии, который предоставляет обширную базу данных изображений и информации о кожных заболеваниях. Если говорить коротко, то данный ресурс крайне схож с ISIC Archive. Он имеет схожие преимущества и недостатки, а также используется в исследованиях в области глубокого обучения и нейронных сетей для диагностики кожных заболеваний.

Kaggle — это популярная онлайн-платформа для специалистов по данным, практиков машинного обучения. Она предоставляет широкий спектр инструментов, ресурсов и наборов данных. В качестве его главного плюса можно выделить широкое разнообразие данных, в том числе уже размеченных. Однако далеко не все данные подходят под решение реальных задач.

Roboflow Universe — это крупнейшая коллекция открытых наборов данных и моделей для компьютерного зрения. Платформа предоставляет доступ к более чем 350 миллионам изображений, 500 000 наборов данных и даже тысячам настроенных моделей. Данных ресурс схож с вышеупомянутым

Kaggle, однако ещё одним его достоинством является удобные API для интеграции моделей в приложения и сервисы.

В ходе выполнения решения данной задачи все эти ресурсы были использованы для поиска исходных данных. ISIC Archive и DermNet имеют обширную базу для исходных данных. Однако у данных ресурсов есть ключевой недостаток - отсутствие готовых датасетов с выполненной разметкой. Данный факт говорит о том, что если использовать исходные данные с этих ресурсов, то придется размечать исходные данные самостоятельно, как и находить высококачественные изображения, или изображения без артефактов и сильного шума.

Чтобы получить датасет с минимальной подготовкой данных и их предобработкой пришлось обратиться к ресурсам Kaggle и Roboflow Universe. В ходе проведенного исследования был выбран датасет с ресурса Roboflow Universe, показанный на рисунке 4.1.

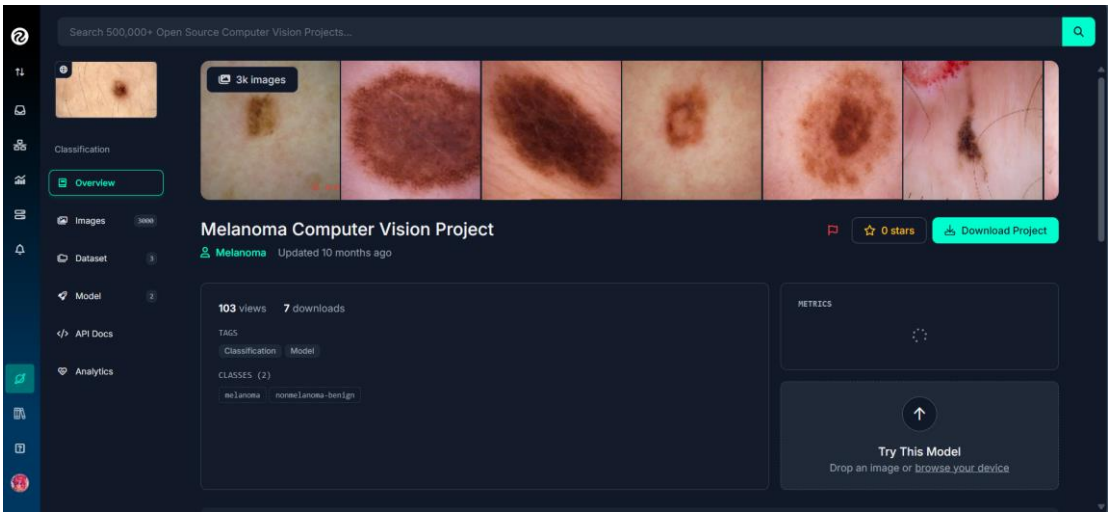


Рисунок 4.1 – датасет меланомы [11]

В качестве достоинств можно отметить неплохой объем данных, а именно 7 300 изображений с учетом проведенной аугментации. Качество данных находится на удовлетворительном уровне. Каждое изображение имеет разрешение 640x640 пикселей. Проведенная в этом датасете аугментация дает по три изображения с горизонтальными переворотами, поворотами на 90 градусов и приближением изображения до 25%. На данной странице представлено 3 версии данного датасета, для выполнения задачи была взята последняя версия под номером 6.

4.1.2 Предварительная обработка

После поиска подходящего датасета надо приступать к процессу предобработки. Даже несмотря на то, что она уже частично выполнена создателем данного датасета, этого недостаточно. Чтобы выполнить предварительную обработку нужно проделать операции нормализации и аугментации.

Процесс нормализация необходим для стабильности и производительности процесса обучения модели. Как уже упоминалось исходные изображения имеют разрешение 640x640 пикселей. С таким разрешением для тренировочного множества процесс обучения будет весьма трудоемким, вследствие чего высока вероятность нехватки вычислительных ресурсов. Также изображения имеют трёхцветный формат со значениями в диапазоне от 0 до 255. Данный диапазон пикселей необходимо нормализировать и привести тем самым к диапазону от 0 до 1, чтобы также повысить стабильность обучения.

Также необходимо самостоятельно провести процесс аугментации, то есть выполнить случайные преобразования для входных изображений: повороты, масштабирование, сдвиги, изменение яркости и контраста. Выполнение данного процесса необходимо сделать с целью увеличения объема данных и улучшения генерализации обучаемой модели.

Процесс разделения данных на тренировочную, валидационную и тестовую выборки также крайне важен для обучения ИНС. Тренировочная выборка, как следует из названия, необходима непосредственно для обучения модели. С помощью данной выборки модель будет настраивать свои параметры таким образом, чтобы минимизировать показатель ошибки. Валидационная выборка нужна для настройки гиперпараметров модели, чтобы впоследствии выбрать наилучшую модель. Также данная выборка поможет предотвратить процесс переобучения. Переобучение – это ситуация, в которой модель слишком хорошо заучивает тренировочные данные, показывая поразительные результаты, но на неизвестных ей данных результаты, напротив, плохие. Тестовая выборка будет использоваться непосредственно для окончательной оценки модели, определения её метрик и определения объективной оценки общей эффективности. Данный процесс выполнять нет полной необходимости в нашем случае. Датасет уже разделен на 3 выборки, нужно будет создать отдельные генераторы для предобработки каждой из выборок.

4.2 Рассматриваемые архитектуры нейросетей

Для решения задачи по классификации кожных новообразований были взяты за основу следующие архитектуры ИНС:

- EfficientNet;
- ResNet;
- YOLO.

Далее будет небольшое пояснение по каждой из данных архитектур.

4.2.1 Семейство EfficientNet

EfficientNet представляет семейство моделей, построенных на основе метода масштабирования, который равномерно масштабирует глубину, ширину и разрешение изображения. EfficientNet – это целое семейство моделей, ниже приведены примеры некоторых из них:

- EfficientNet-B0: базовая модель с оптимальной производительностью на малых вычислительных ресурсах;
- EfficientNet-B3: более глубокая модель с улучшенной точностью;
- EfficientNet-B7: самая мощная версия, обладающая максимальной точностью при повышенных затратах вычислений.

Всего будет рассмотрено 8 моделей данного семейства от B0 до B7. Как уже понятно, всех их объединяет общая идея. А в качестве различия в данных моделях выступает их сложность и количество параметров, как и следствие размер модели, время обучения и точность. Более простые модели можно применить к простым данным и в случаях, когда важна скорость. Более сложные модели можно применить к более сложным данным и в областях, где большую значимость имеет точность. Но надо быть осторожным, ведь сложные модели часто склонные к переобучению.

Выбор пал на данное семейство из-за высокого показателя точности для оказываемой нагрузки. По сравнению с архитектурами, имеющими такую же скорость работы, архитектура EfficientNet является более легковесной. В то же время по сравнению с более быстро действенной архитектурой, такой как MobileNet, EfficientNet показывает лучшие результаты в аспекте точности. Сравнить наглядно характеристики архитектур можно в приложении А.

Основы этих моделей можно получить в модуле keras.applications, который является частью библиотеки Tensorflow.

4.2.2 Архитектура ResNet

До появления ResNet было известно, что глубокие нейронные сети теоретически способны моделировать сложные зависимости в данных. Однако на практике увеличение глубины часто приводит к ухудшению производительности модели, несмотря на использование методов, таких как нормализация и регуляризация. Основная причина этого — затухание градиента, затрудняющее обновление параметров на нижних слоях сети. Именно архитектура ResNet оказалась способной решить проблему исчезающего градиента с помощью остаточных соединений.

В качестве плюсов данной архитектуры можно выделить высокую сходимость и высокую производимость.

Как и в случае EfficientNet, здесь существует целое семейство архитектур с одной идеей, но разными конфигурациями. В данной работе будут рассмотрены архитектуры ResNet-50v2, ResNet-101v2 и ResNet-151v2**:** "глубокие" версии с 50, 101 и 152 слоями, использующие расширенные остаточные блоки.

4.2.3 Семейство моделей YOLO

YOLO — это семейство моделей глубокого обучения. Изначально данная модель применялась для решения задачи детекции объектов на изображении. Однако существуют версии моделей, которые способны также решать задачи классификации и сегментации. Именно поэтому в данной работе будут рассматриваться YOLOv8 и YOLOv11. Именно в этих версиях существует возможность решать данные задачи. Также проверим, какая версия покажет себя лучше, более новая 11 версия, или же старая версия под номером 8.

Из главных преимуществ данной архитектуры можно выделить высокую скорость работы, что может позволить использовать данную архитектуру в системах, работающих в реальном времени. А также простота использования данной архитектуры может сыграть важную роль в вопросе интеграции в другие системы.

4.3 Процесс обучения

Как уже упоминалось данные разделены на три выборки, каждая из которых играет важную роль в процессе обучения. Затем для нашей

собственной предобработки воспользуемся объектом ImageDataGenerator, предоставляемым библиотекой TensorFlow. С помощью данного объекта мы можем нормализировать данные, а также указать различные параметры для поворотов, сдвигов, масштабирования изображений. Таким образом мы увеличиваем объем наших выборок. Процесс аугментации проводится только для тренировочной выборки. Для валидационной и тестовой мы проводим только нормализацию — деление значений пикселей изображения на 255, чтобы их значения были в диапазоне от 0 до 1.

Модели мы используем предобученные на датасете ImageNet из всё той же библиотеки TensorFlow. Очевидно, этого будет недостаточно, так как ImageNet не специализированный датасет. Но факт того, что модель уже предобучена может облегчить процесс обучения. Верхние слои в модель не включаем и указываем размер входных данных. Таким образом мы получаем базовую модель. Её проблема в том, что она ориентирована на классы датасета ImageNet. Чтобы это исправить добавляем к базовой модели слой глобального среднего пулинга для двумерных данных и полносвязный слой с сигмоидной функцией активации – меланома/не меланома.

В качестве оптимизатора моделей из TensorFlow указываем оптимизатор Adam с адаптивной скоростью обучения. А метрикой, на которую будет ориентироваться процесс обучения, указываем точность.

Так же для достижения максимальной точности применяем следующие методы:

- Grid Search**:** перебор параметров;
- Learning Rate Scheduler: адаптивная настройка шага обучения.

Модели YOLO берутся из библиотеки Ultralycits. Для этого просто импортируем объект YOLO и при его создании задаем в качестве параметра имя модели. К примеру, подобная модель “yolo11n-cls.pt”, где 11 является версией модели, n – это размер модели, в данном случае самый маленький, и cls, что указывает на решение задачи классификации.

4.4 Анализ и сравнение результатов

После изучения моделей ИНС на подготовленных данных были произведены оценки каждой из архитектур и составлена соответствующая таблица. Данную таблицу можно посмотреть полностью в приложении Б. Ниже, в таблице 4.1, представлены результаты некоторых разных архитектур для сравнения.

Таблица 4.1 – оценка трех различных архитектур ИНС различными метриками

модель\метрики оценки	Accuracy	F1-Score	ROC-AUC	Время оценки (сек. на 1 изображение)	Precision (melanoma)	Recall (melanoma)	Recall (melanoma)	Recall (nonmelanoma)
EfficientNet-B3	0.80	0.82	0.87	0.39	0.88	0.63	0.70	0.94
ResNet101v2	0.83	0.85	0.90	0.47	0.90	0.67	0.73	0.94
YOLOv8	0.78	0.80	0.84	0.33	0.86	0.58	0.68	0.93

По данной таблице и приложению Б можно заключить следующие выводы:

- модель YOLO показала превосходящие результаты в скорости по сравнению с другими моделями. Однако данная архитектура уступает более сложным моделям в точности классификации;

- модель EfficientNet начиная с версии B4 сравнима с архитектурой ResNet, модели с номерами B3 и меньше отстают в метриках оценки качества классификации, но превосходят их в скорости;

- все модели показали лучшие показатели Precision для меланомных случаев. Ситуация по показателю Recall – противоположная.

4.5 Выводы

В результате данной главы был проведен полностью процесс обучения и в конечном итоге получены модели, способные на решение задачи классификации меланомных новообразований с достаточно высокой точностью и возможностью интеграции в целевую систему.

ГЛАВА 5
РЕАЛИЗАЦИЯ ЦЕЛЕВЫХ КОМПОНЕНТОВ СИСТЕМЫ

5.1 База данных

В ходе выполнения данной работы было определена необходимость наличия базы данных. Данная база должна будет содержать данные о пациентах, врачах, медицинских учреждениях, результатах анализа цифровых изображений искусственной нейронной сетью и сообщений. На рисунке 5.1 продемонстрирована ER-диаграмма базы данных текущей версии.

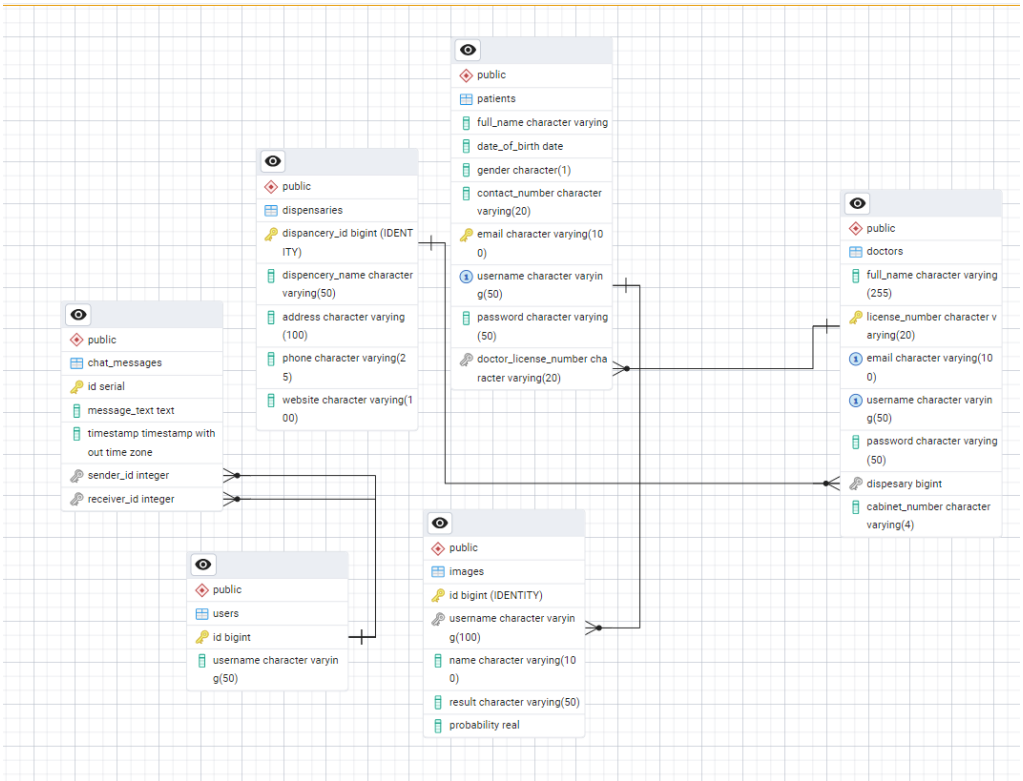


Рисунок 5.1 – ER-диаграмма базы данных

Таблица users необходима для объединения врачей и пациентов как пользователей данной системы, к тому же данная таблица нужна для хранения сообщений между двумя пользователями. Именно на уникальный идентификатор пользователей из этой таблицы будут ссылаться внешние ключа отправителя и получателя из таблицы messages.

Таблицы – doctors, patients хранят информацию о врачах и пациентах. Последние на текущем этапе достаточно типичные и похожи на любую таблицу с данными пользователей.

Таблица images хранит результаты первичной диагностики нейронной сети в виде имени файла цифрового изображения, которое последнее получает в момент отправки на сервер, анализа искусственной нейронной сетью, вероятности истинности полученного результата и имени пользователя,

который отправил фотографию на сервер. Сами изображения хранятся на физическом устройстве, на котором запущен сервер и их можно будет идентифицировать по имени файла, состоящем из никнейма пользователя, дате и времени отправки.

5.2 Серверный компонент

Для серверного компонента на текущем этапе нам нужно создать три модуля и один файл конфигурации. Данный файл выглядит следующим образом:

```
DB_CONFIG = {
    'dbname': 'rds',
    'user': 'postgres',
    'password': 'ProjectPass123',
    'host': 'localhost',
    'port': '5432'
}
```

В данном файле содержится информация, с помощью которой сервер сможет получать доступ к базе данных: имя базы данных, пользователь, пароль, хост и порт.

В следующий файл запишем метод, который будет возвращать пользователя из таблицы пациентов по его никнейму, а также метод, который будет возвращать объект типа connection:

```
def get_db_connection():
    conn = psycopg2.connect(**DB_CONFIG)
    return conn
```

Последний будет в завершенной целевой системе использоваться довольно часто, так как при любом взаимодействии с базой данных необходимо установить с ней соединение. Полный исходный текст данного модуля можно увидеть в приложении В.

Следующий файл содержит класс-нейросеть, который при инициализации объекта класса загружает искусственную нейронную сеть из заранее заготовленного файла весов обученной нейронной сети. Класс также содержит метод, который принимает в качестве аргумента строку, являющуюся путем к файлу цифрового изображения, после чего выполняет загрузку и подготовку изображения к анализу и выполняет предсказание одного из двух классов. В конце метод возвращает объект типа кортеж, содержащий номер предсказанного класса и вероятность принадлежности к этому классу, измеряемую от 0 до 1. Полный исходный текст данного модуля можно посмотреть в приложении Г.

Последний модуль является основным файлом, в котором и происходит запуск сервера и методы, выполняющиеся при получении того или иного http-запроса. Всего реализованы следующие функции с соответствующими эндпоинтами:

- *login()* выполняется при запросе на аутентификацию пользователя пациента, в ней проверяется наличие данного пользователя в базе данных, корректность имени пользователя и пароля, а также выдача токена доступа;

- *login_doctor()* – аналогичен предыдущему, но применяется по отношению к врачам;

- *classify_image()* выполняется при запросе от клиента на первичную диагностику по цифровому изображения от ИНС. Метод получает из https-запроса файл изображения, передает его классу нейронной сети, сохраняет изображение локально на сервере и сохраняет результаты анализа в базе данных, а в конце отправляет результаты первичной диагностики клиенту;

- *download_image(image_name)* данный метод отправляет фотографию клиенту при получении соответствующего http-запроса. Данный метод позволит врачу просматривать цифровые изображения;

- *get_patient_list(license_number)* – возвращает список пациентов, приписанных конкретному врачу;

- *get_image_list(email)* – возвращает список изображений и результатов первичной диагностики. То есть данные из таблицы images для конкретного пациента. С помощью этого списка врач сможет получать конкретные цифровые изображения и результаты первичной диагностики посредством ИНС для данного изображения;

- методы *send_message()* и *chat_history()* позволяют соответственно отправлять сообщения другому пользователю и загружать историю сообщений. Что в совокупности реализует процесс обмена сообщениями;

Полный исходный текст главного модуля сервера можно увидеть в приложении Д.

5.3 Компонент мобильного клиента

Клиентскую часть для пациентов, как уже было упомянуто ранее, было решено делать для операционной системы Android в виде мобильного приложения. Ключевыми компонентами приложения являются *Activity*.

Activity – это один из ключевых компонентов android-приложений, представляющий собой отдельный экран со своим пользовательским интерфейсом. Всего нам нужны будут три *Activity*. Первая будет представлять собой экран авторизации, вторая – интерфейс для загрузки изображения на сервер и отображения полученного результата от сервера. Третья будет

представлять интерфейс для обмена сообщениями дистанционной консультации. Исходный текст данных компонентов можно увидеть в приложении Е.

Далее нужно создать ещё два модуля-сервиса, которые будут отправлять http-запросы на сервер для завершения авторизации, обмена сообщениями и получения результатов анализа искусственной нейронной сети. Однако перед этим нужно изменить файл манифеста. В этом файле также установить в качестве стартового экрана *Activity*, отвечающей за авторизацию:

```
<activity
    android:name=".ui.StartActivity"
    android:exported="true"
    android:label="@string/title_activity_start"
    android:theme="@style/Theme.MobileClient">
    <intent-filter>
        <action android:name="android.intent.actionMAIN" />
        <category
    android:anme="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Затем устанавливает три разрешения:

```
<uses-permission
android:name="android.permission.READ_MEDIA_IMAGES" />
<uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.INTERNET" />
```

Первые два разрешения дадут приложению доступ для чтения медиа файлов и внешнего хранилища. Это необходимо для того, чтобы была возможность выбрать изображение, которое хранится на устройстве, для отправки.

Третье разрешение необходимо для работы с интернетом. В нашем случае без него у приложения не будет возможности отправлять http-запросы и получать ответы на них.

Первый сервис содержит функцию, которая отправляет http-запрос, содержащий имя пользователя и пароль в хешированном виде. Функция затем получает ответ, содержащий либо текст ошибки, либо токен доступа. Также есть реализованный функционал запросов-ответов для обмена сообщениями.

Сервис отправки изображений отправляет http-запрос с файлом изображения и хешированным именем пользователя, впоследствии получает ответ с диагнозом и вероятность истинности диагноза. Перед отправкой сервис также предварительно преобразует любое изображение в файл с

расширением jpg и именем, содержащем имя пользователя, дату и время отправки.

Исходный текст двух сервисов можно увидеть в приложении Ж.

5.4 Компонент десктопного клиента

Как уже упоминалось, клиент для врачей будет представлять собой десктопное приложение, в проектировании которого нам поможет библиотека Tkinter.

Данные клиент по функционалу можно разделить на два модуля:

- модуль взаимодействия с сервером;
- модуль построения визуального интерфейса.

В модуле взаимодействия с сервером создаем соответствующий класс с необходимыми методами для взаимодействия с серверной частью. Главными методами являются:

- *md_5encode(x: str)* – хеширование данных при авторизации;
- *login_doctor(username: str, password: str)* – авторизация пользователя-врача;
- *get_patients_by_license(license_number)* – получение списка пациентов врача по номеру его лицензии;
- *get_image_list(email)* – получение списка имен изображений с предварительным диагнозом от ИНС;
- *get_image(image_name)* – получение непосредственно самого изображения для его просмотра;
- *get_chat_message()* - получение истории сообщений;
- *send_chat_message()* – отправка сообщения пациенту.

В качестве модуля построения визуального интерфейса создаем собственный класс, который при инициализации создает все необходимые интерфейсы и реализует логику их взаимодействия между собой. На рисунке 5.2 показан главный экран данного клиента.

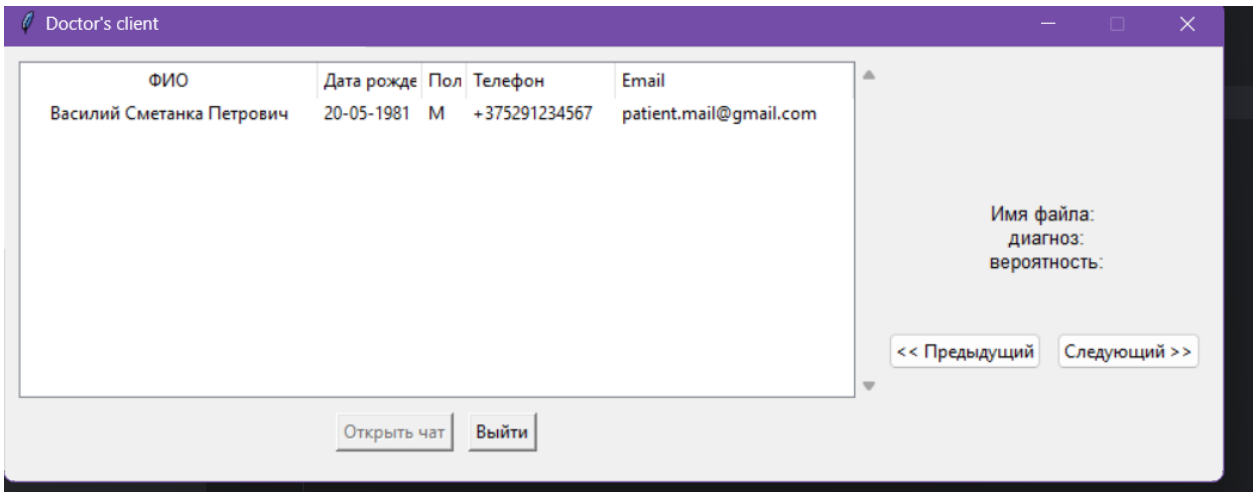


Рисунок 5.2 – основной экран клиента для врачей

Здесь присутствует таблица со списком пациентов врача, где после выбора пациента по левому клику врач сможет просматривать справа изображения, которые пациент отправлял для первичной диагностики посредством искусственной нейронной сети. Также врачу после выбора пациента станет доступна кнопка, открывающая чат с конкретным пациентом.

Исходный текст модулей окна и сервиса можно увидеть в приложениях И и К соответственно.

5.5 Выводы

В данной главе представлены основные моменты, связанные с разработкой базы данных, клиента пациентов, клиента для врачей и сервера, необходимые для разработки целевой системы дистанционной диагностики. Был показан список функций, которые были реализованы для целевой системы. Объяснены отдельные моменты работы созданных компонентов. Показан главный экран клиента для врачей.

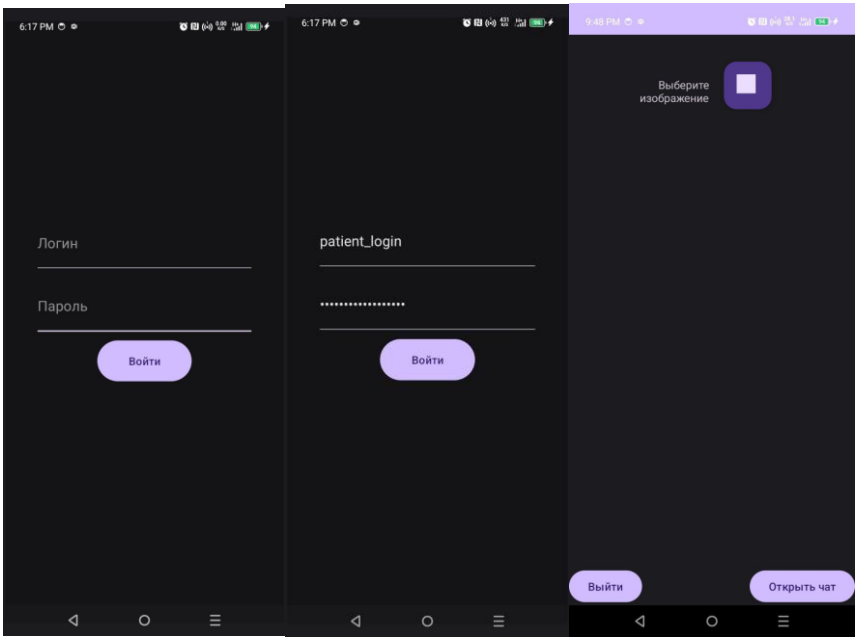
ГЛАВА 6
РЕШЕНИЕ ПРИКЛАДНОЙ ЗАДАЧИ

6.1 Постановка задачи

Смоделируем следующую ситуацию. Мы - пациент, у которого есть на телефоне установленный клиент приложения. Мы уже зарегистрированы в системе. У нас на теле есть родинка, которая может нас беспокоить. Но мы не можем в ближайшее время записаться на прием к врачу, поскольку на данный момент мы уехали далеко в сельскую местность. Но устранить опасения хочется, поэтому мы воспользуемся функцией первичной диагностики от искусственной нейронной сети, которая может распознать заболевание кожи.

6.2 Решение задачи

Запускаем приложение и видим экран, как на рисунке 6.1 (а). Затем вводим свои данные и нажимаем кнопку “Войти”. Мы попадаем на экран, показанный на рисунке 6.1 (в).



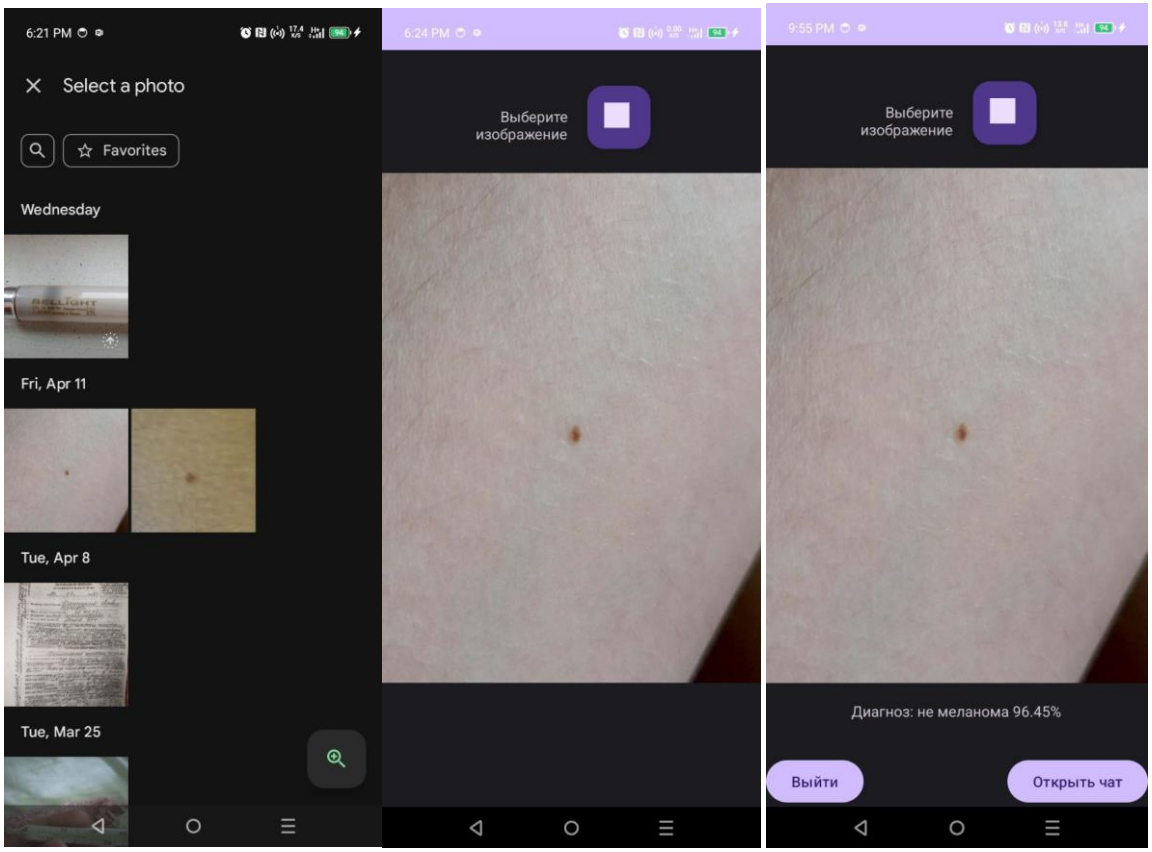
а б в

а - стартовый экран авторизации; б - экран с заполненными полями; в - экран предварительной диагностики от ИНС

Рисунок 6.1 – стартовый экран

Теперь нажимаем кнопку “Выберите изображение”, после чего нам предложат выбрать одно из нескольких приложений, которые уже есть на телефоне и используются для просмотра галереи. Выбираем любое и попадаем

в галерею, как на рисунке 6.2 (а). Выбираем заранее сделанную нами фотографию родинки и ожидаем ответа от сервера. В это время экран выглядит как на рисунке 6.2 (б).

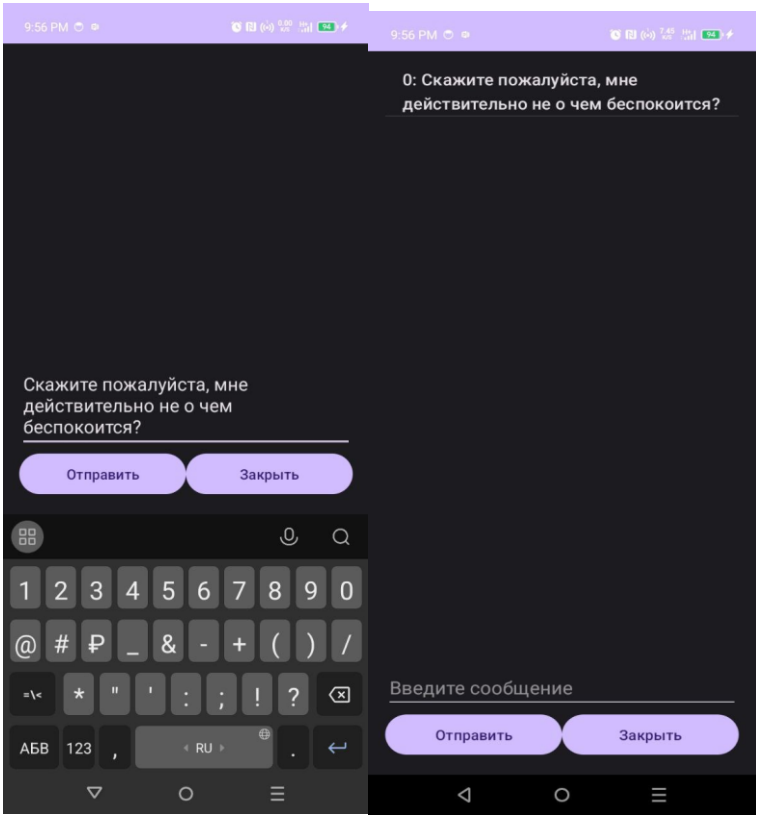


а б в

а - галерея; б - экран во время ожидания ответа от сервера; в - экран с результатом, который был получен от сервера

Рисунок 6.2 – главный экран

После получения ответа от сервера мы получаем результат в 96.45%, что изображение родинки на фотографии не является злокачественным образованием меланомы, как это видно на рисунке 6.2 (в). Учитывая такую большую вероятность, можно отбросить беспокойства о происхождении данной родинки. Но мы считаем, что не помешает написать врачу. Нажимаем кнопку “Открыть чат” и попадаем на экран, показанный на рисунке 6.3 (а) и пишем сообщение. На рисунке 6.3 (б) продемонстрирован экран с уже отправленным сообщением.



а б

а - экран с отправкой сообщения; б - экран после отправки сообщения

Рисунок 6.3 – экран с чатом

Теперь переходим на другую сторону. Врач открывает приложение и видит окно, показанное на рисунке 6.4, вводит свои данные и затем нажимает кнопку вход.

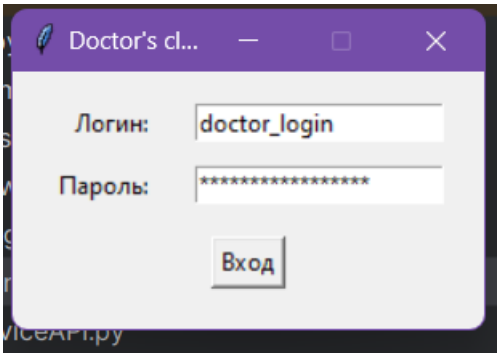


Рисунок 6.4 – окно авторизации десктопного клиента

После входа врач попадает на главное окно, как на рисунке 6.5. Затем щелчком левой кнопки мыши по пациенту из таблицы он получает то самое цифровое изображение, которое мы отправляли для первичной диагностики посредством ИНС (смотри рисунок 6.6).

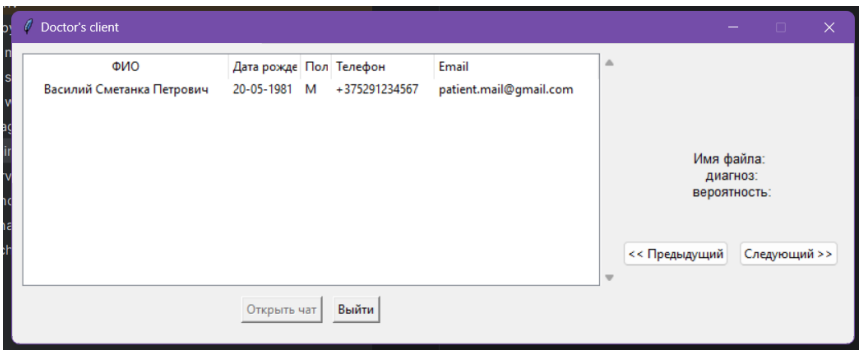


Рисунок 6.5 – главное окно

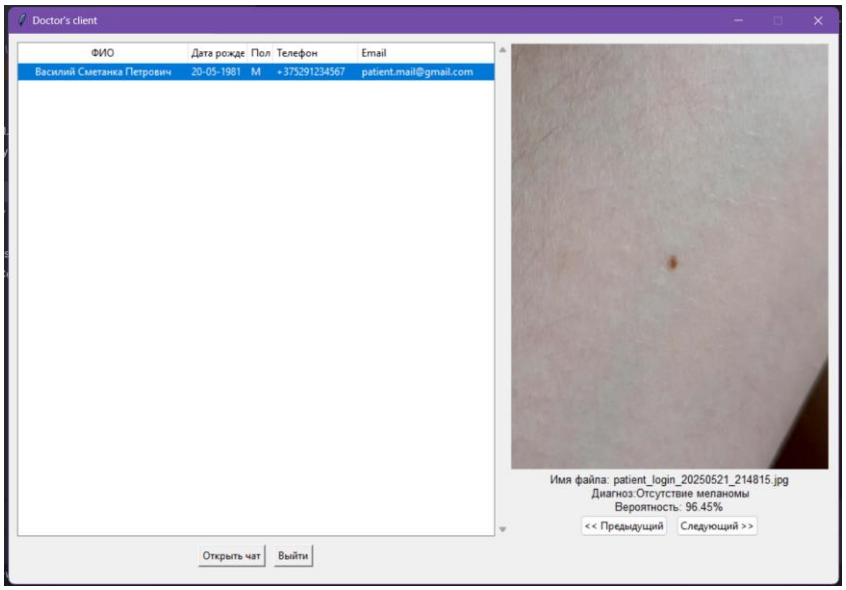


Рисунок 6.6 – главное окно после выбора пациента

Просмотрев диагноз от ИНС и лично не найдя ничего подозрительного на изображении, врач открывает чат и видит то, что изображено на рисунке 6.7

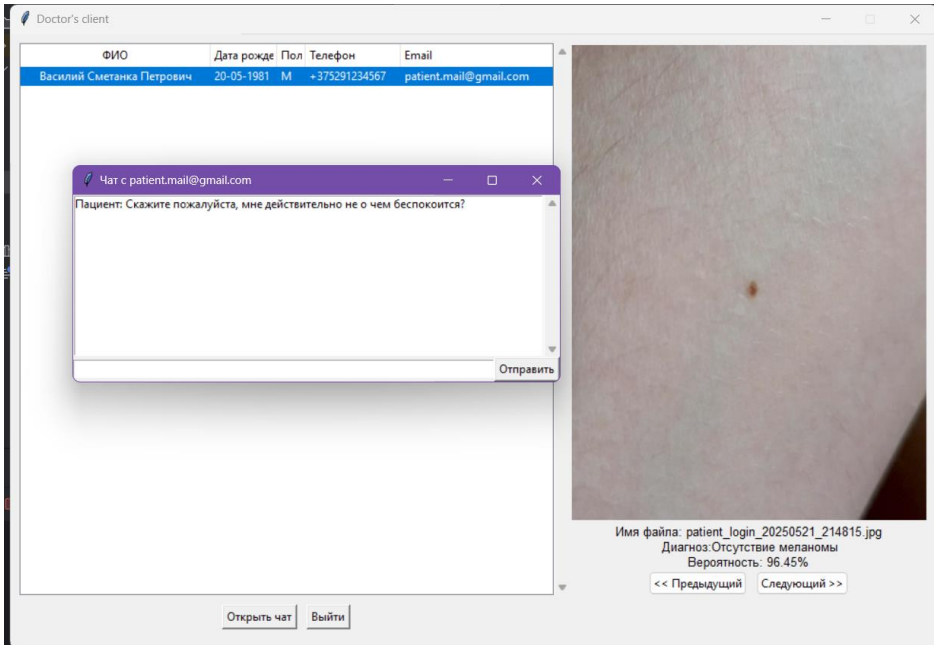
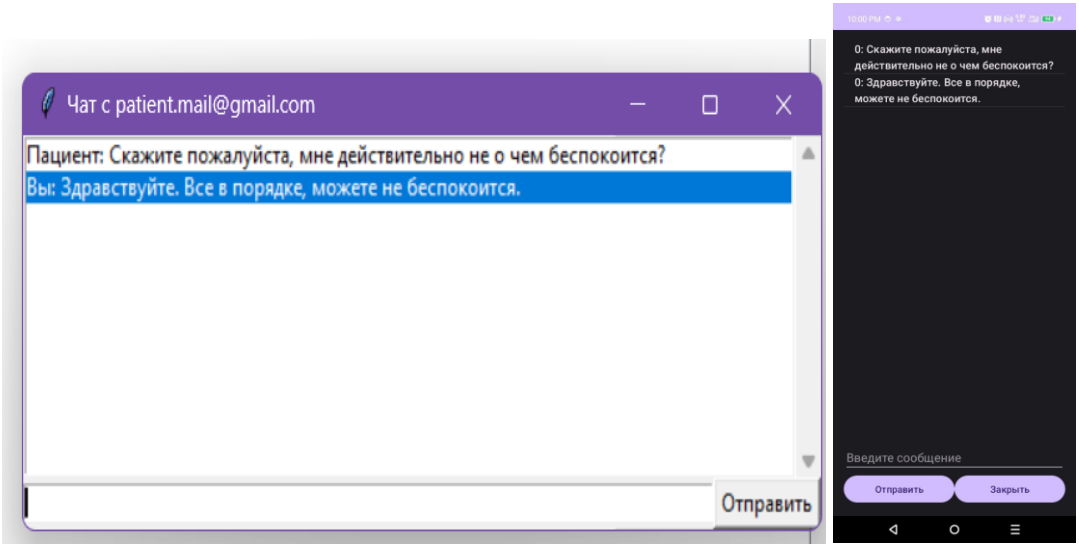


Рисунок 6.7 – открытый чат с пациентом

После врач пишет сообщение пациенту о своем заключении. Как это выглядит для врача показано на рисунке 6.8 (а), у пациента после получения сообщения экран выглядит как на рисунке 6.8 (б).



а б

а - история сообщение со стороны врача; б - история сообщений со стороны пациента

Рисунок 6.8 – экран с чатом у обоих клиентов

6.3 Выводы

В данной главе смоделирована ситуация и продемонстрирована корректная работа компонента системы, с помощью которого пользователь получил предварительную диагностику кожного новообразования от нейронной сети с положительным результатом на отсутствие меланомы. Для этого он воспользовался камерой своего телефона и мобильным клиентом целевой системы.

Впоследствии было показано, как все выглядит со стороны врача и продемонстрирован простой и удобный процесс дистанционной диагностики.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы были решены основные задачи по проектированию и реализации целевой системы дистанционной диагностики, предварительно исследована предметная область. В том числе была проделана следующая работа:

- успешно обучена искусственная нейронная сеть, решающая задачу классификации злокачественных новообразований кожи;
- продемонстрирован конкретный пример использования целевой системы для предварительной дистанционной диагностики от искусственной нейронной сети на предмет присутствия меланомы на изображении;
- реализованы клиентские и серверная части целевой системы;
- спроектирована база данных;
- определён наиболее подходящий вариант решения поставленной задачи;
- выбран инструментарий для реализации целевой системы;
- углублены знания по разработке клиент-серверных приложений на языке Python, android-разработке на языке программирования Kotlin.

По итогу задание на дипломную работу выполнено: реализована целевая система по дистанционной диагностике злокачественных новообразований и продемонстрирована её эффективность.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Айгнер, Себастьян; Елизаров, Роман; Жемеров, Дмитрий; Исакова, Светлана. Kotlin в действии / С. Айгнер, Р. Елизаров, Д. Жемеров, С. Исакова. – 2-е изд. – СПб.: Питер, 2025. – 560 с.
2. Всемирная организация здравоохранения. Ультрафиолетовое излучение [электронный ресурс]. Режим доступа: https://www.who.int/ru/news-room/fact-sheets/detail/ultraviolet-radiation. - Дата доступа: 14.04.2025
3. Шолле, Франсуа. Глубокое обучение на Python / Франсуа Шолле. – 2-е изд. – СПб.: Питер, 2023. – 576 стр.
4. American Cancer Society. How to Spot Skin Cancer [Electronic resource] – Mode of access: https://www.cancer.org/cancer/latest-news/how-to-spot-skin-cancer.html. Date of access: 14.04.2025
5. American Medical Association. Telehealth Implementation Playbook [Electronic resource]. – Mode of access: https://www.ama-assn.org/delivering-care/public-health/ama-telehealth-implementation-playbook. – Date of access: 16.04.2025.
6. Bashshur R.L., et al. The empirical Foundations of Telemedicine Interventions in Primary Care [Electronic resource]. – Mode of access: https://doi.org/10.1089/tmj.2016.0045. – Date of access: 19.04.2025.
7. Flask Documentation [Electronic resource]. – Mode of access: https://flask.palletsprojects.com/. – Date of access: 17.04.2025.
8. Flask-JWT-Extended Documentation [Electronic resource]. – Mode of access: https://flask-jwt-extended.readthedocs.io/. – Date of access: 17.04.2025.
9. Institute of Medicine. The Role of Telehealth in an Evolving Health Care Environment [Electronic resource]. – Mode of access: https://www.ncbi.nlm.nih.gov/books/NBK207145/. – Date of access: 15.04.2025.
10. McKinsey & Company. Telehealth: A Post-COVID-19 Reality [Electronic resource]. – Mode of access: https://www.mckinsey.com/industries/healthcare/our-insights/telehealth-a-post-covid-19-reality. – Date of access: 15.04.2025.
11. Melanoma Dataset [Electronic resource]. — Roboflow. - Mode of access: https://universe.roboflow.com/melanoma-mwh97/melanoma-xrygq. – Date of access: 05.10.2024.
12. Moore, Alan D. Python GUI Programming with Tkinter / Alan D. Moore. – 2nd ed. – Birmingham: Packt Publishing, 2021. – 644 p.
13. Totten, A.M. et al. Telehealth: Mapping The Evidence for Patient Outcomes From Systematic Reviews [Electronic resource]. – Mode of access: https://effectivehealthcare.ahrq.gov/products/telehealth/technical-brief. – Date of access: 16.04.2025.

14. WHO Global Observatory for eHealth. Telemedicine: Opportunities and Developments in Member States [Electronic resource]. – Mode of access: https://apps.who.int/iris/handle/10665/44497. – Date of access: 15.04.2025.

ПРИЛОЖЕНИЕ А

ХАРАКТЕРИСТИКИ АРХИТЕКТУР ИНС

Таблица А.1 Характеристики архитектур ИНС, предоставляемых keras.application

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6
InceptionV3	92	77.9%	93.7%	23.9M	189	42.2	6.9
InceptionResNetV2	215	80.3%	95.3%	55.9M	449	130.2	10.0
MobileNet	16	70.4%	89.5%	4.3M	55	22.6	3.4
MobileNetV2	14	71.3%	90.1%	3.5M	105	25.9	3.8
DenseNet121	33	75.0%	92.3%	8.1M	242	77.1	5.4
DenseNet169	57	76.2%	93.2%	14.3M	338	96.4	6.3
DenseNet201	80	77.3%	93.6%	20.2M	402	127.2	6.7
NASNetMobile	23	74.4%	91.9%	5.3M	389	27.0	6.7
NASNetLarge	343	82.5%	96.0%	88.9M	533	344.5	20.0
EfficientNetB0	29	77.1%	93.3%	5.3M	132	46.0	4.9
EfficientNetB1	31	79.1%	94.4%	7.9M	186	60.2	5.6
EfficientNetB2	36	80.1%	94.9%	9.2M	186	80.8	6.5
EfficientNetB3	48	81.6%	95.7%	12.3M	210	140.0	8.8
EfficientNetB4	75	82.9%	96.4%	19.5M	258	308.3	15.1
EfficientNetB5	118	83.6%	96.7%	30.6M	312	579.2	25.3
EfficientNetB6	166	84.0%	96.8%	43.3M	360	958.1	40.4
EfficientNetB7	256	84.3%	97.0%	66.7M	438	1578.9	61.6
EfficientNetV2B0	29	78.7%	94.3%	7.2M	-	-	-
EfficientNetV2B1	34	79.8%	95.0%	8.2M	-	-	-
EfficientNetV2B2	42	80.5%	95.1%	10.2M	-	-	-
EfficientNetV2B3	59	82.0%	95.8%	14.5M	-	-	-
EfficientNetV2S	88	83.9%	96.7%	21.6M	-	-	-
EfficientNetV2M	220	85.3%	97.4%	54.4M	-	-	-
EfficientNetV2L	479	85.7%	97.5%	119.0M	-	-	-

ПРИЛОЖЕНИЕ Б

РЕЗУЛЬТАТЫ ОЦЕНКИ ИНС

Таблица Б.1 - показатели метрик Accuracy, F1-Score, ROC-AUC и времени оценки моделей, полученные в процессе обучения.

модель	accuracy	F1-score	ROC-AUC	время оценки (сек. на 1 изображение)
EfficientNet-B0	0.75	0.78	0.83	0.35
EfficientNet-B1	0.77	0.80	0.85	0.37
EfficientNet-B2	0.79	0.81	0.86	0.38
EfficientNet-B3	0.80	0.82	0.87	0.39
EfficientNet-B4	0.82	0.84	0.89	0.42
EfficientNet-B5	0.83	0.85	0.90	0.45
EfficientNet-B6	0.84	0.86	0.91	0.48
EfficientNet-B7	0.85	0.86	0.92	0.50
ResNet-50v2	0.81	0.83	0.88	0.42
ResNet-101v2	0.83	0.85	0.90	0.47
ResNet-151v2	0.85	0.87	0.92	0.53
YOLOv8	0.78	0.80	0.84	0.33
YOLOv11	0.83	0.85	0.89	0.40

Таблица Б.2 - показатели метрик Precision и Recall для обоих классов, полученные в процессе обучения.

модель	precision (melanoma)	recall (melanoma)	precision (nonmelanoma)	recall (nonmelanoma)
EfficientNet-B0	0.85	0.55	0.65	0.92
EfficientNet-B1	0.86	0.58	0.68	0.93
EfficientNet-B2	0.87	0.60	0.69	0.94
EfficientNet-B3	0.88	0.63	0.70	0.94
EfficientNet-B4	0.89	0.65	0.72	0.95
EfficientNet-B5	0.90	0.67	0.73	0.95
EfficientNet-B6	0.91	0.68	0.74	0.96
EfficientNet-B7	0.92	0.70	0.75	0.96
ResNet-50v2	0.89	0.65	0.72	0.93
ResNet-101v2	0.90	0.67	0.73	0.94
ResNet-151v2	0.91	0.70	0.75	0.95
YOLOv8	0.86	0.58	0.68	0.93
YOLOv11	0.89	0.68	0.74	0.94

ПРИЛОЖЕНИЕ В

ПОЛНЫЙ ЛИСТИНГ МОДУЛЯ, ВЗАИМОДЕЙСТВУЮЩЕГО С БАЗОЙ ДАННЫХ

```
from datetime import datetime
from typing import List, Dict

import psycopg2
from config import DB_CONFIG

def get_db_connection():
    conn = psycopg2.connect(**DB_CONFIG)
    return conn

def get_user(username, table):
    connection = None
    result = None
    try:
        connection = get_db_connection()
        cur = connection.cursor()

        cur.execute(f"SELECT * FROM {table} where username = %s;",
(username,))
        result = cur.fetchall()
        connection.commit()

    except Exception as _ex:
        print("[INFO] Error while working with PostgreSQL", _ex)
    finally:
        if connection:
            connection.close()
            print("[INFO] PostgreSQL connection closed")
        if 'cur' in locals():
            cur.close()
        return result

def get_patients(license_number):
    connection = None
    result = None
    try:
        connection = get_db_connection()
        cur = connection.cursor()
        cur.execute(f"SELECT * FROM patients where doctor_license_number =
%s;", (license_number,))
        result = cur.fetchall()
        connection.commit()
    except Exception as _ex:
        print("[INFO] Error while working with PostgreSQL", _ex)
    finally:
        if connection:
            connection.close()
            print("[INFO] PostgreSQL connection closed")
        if 'cur' in locals():
            cur.close()
        if len(result) != 0:
            edited_result = []
            for i in range(len(result)):
                FIO = result[i][0].replace('_', ' ')
                birth_date = ""
                if result[i][1].month < 10:
```

```
                    birth_date = str(result[i][1].day) + "-0" +
str(result[i][1].month) + "-" + str(result[i][1].year)
                else:
                    birth_date = str(result[i][1].day) + "-" +
str(result[i][1].month) + "-" + str(result[i][1].year)
                edited_result.append((FIO, birth_date, result[i][2],
result[i][3], result[i][4]))
            return edited_result
        else:
            return result

def get_image_names(email):
    connection = None
    result = None
    try:
        connection = get_db_connection()
        cur = connection.cursor()
        cur.execute(f"SELECT username FROM patients where email = %s;",
(email,))
        username = cur.fetchall()[0][0]
        cur.execute(f"SELECT * FROM images where username = %s;",
(username,))
        result = cur.fetchall()
        connection.commit()

    except Exception as _ex:
        print("[INFO] Error while working with PostgreSQL", _ex)
    finally:
        if connection:
            connection.close()
            print("[INFO] PostgreSQL connection closed")
        if 'cur' in locals():
            cur.close()
        return result

def save_chat_message(sender_username, receiver_username, message_text):
    connection = None
    try:
        connection = get_db_connection()
        cur = connection.cursor()

        timestamp = datetime.now()

        cur.execute("""
            INSERT INTO chat_messages
            (sender_id, receiver_id, message_text, timestamp)
            VALUES (%s, %s, %s, %s)
            RETURNING id
        """, (sender_username, receiver_username, message_text, timestamp))

        message_id = cur.fetchone()[0]
        connection.commit()
        return message_id

    except Exception as ex:
        print("[INFO] Error saving chat message:", ex)
        return None
    finally:
        if connection:
            connection.close()


def get_chat_messages(user1, user2):
```

```
connection = None
try:
    connection = get_db_connection()
    cur = connection.cursor()

    cur.execute("""
        SELECT * FROM chat_messages
        WHERE (sender_id = %s AND receiver_id = %s)
           OR (sender_id = %s AND receiver_id  = %s)
        ORDER BY timestamp
    """, (user1, user2, user2, user1))

    messages = []
    for row in cur.fetchall():
        messages.append({
            'id': row[0],
            'sender_id': row[3],
            'receiver_id': row[4],
            'text': row[1],
            'timestamp': row[2].isoformat()
        })
    print(messages)
    return messages

except Exception as ex:
    print("[INFO] Error fetching chat messages:", ex)
    return []
finally:
    if connection:
        connection.close()
```

ПРИЛОЖЕНИЕ Г

ПОЛНЫЙ ЛИСТИНГ МОДУЛЯ С ПРЕДОБУЧЕННОЙ ИНС

```
from tensorflow.keras.models import load_model
import tensorflow as tf
from tensorflow.keras.preprocessing import image
import numpy as np

class NN:
    def __init__(self, model_path: str):
        self._model = tf.keras.models.load_model(model_path)

    def predict_single_image(self, image_path):
        """
            Предсказывает класс для одного изображения

            :param image_path: путь к изображению
            :return: предсказанный класс (0 или 1 - не меланома и меланома) и
вероятность
            """
        # Загрузка и подготовка изображения
        img = image.load_img(image_path, target_size=(224, 224))
        img_array = image.img_to_array(img)
        img_array = np.expand_dims(img_array, axis=0)
        img_array /= 255.0

        prediction = self._model.predict(img_array)
        probability = prediction[0][0]
        class_pred = 1 if probability > 0.5 else 0
        probability = prediction[0][0] if class_pred == 1 else 1 -
prediction[0][0]
        return class_pred, probability
```

ПРИЛОЖЕНИЕ Д

ПОЛНЫЙ ЛИСТИНГ MAIN-ФАЙЛА СЕРВЕРА

```
import hashlib
from flask import Flask, request, jsonify, send_file
import os
from flask_jwt_extended import JWTManager, create_access_token, jwt_required,
get_jwt_identity
from skin_nn_classifation import NN
from data_base_helper import get_db_connection, get_user, get_patients,
get_image_names, save_chat_message, \
    get_chat_messages

app = Flask(__name__)
app.config["JWT_SECRET_KEY"] = "my-secret-secret-key"
jwt = JWTManager(app)

# Инициализация модели нейросети
nn = NN(os.path.join(os.getcwd(), "NeuralNetwork", "EfficientNetB3.keras"))

# Настройка папки для загрузки изображений
UPLOAD_FOLDER = 'images'
if not os.path.exists(UPLOAD_FOLDER):
    os.makedirs(UPLOAD_FOLDER)

@app.route('/classify', methods=['POST'])
@jwt_required()
def classify_image():
    """Классификация изображения с помощью нейросети"""
    username = get_jwt_identity()

    if 'file' not in request.files:
        return jsonify({"error": "No file part"}), 400

    file = request.files['file']
    if file.filename == '':
        return jsonify({"error": "No selected file"}), 400

    if file:
        # Сохранение файла
        file_path = os.path.join(UPLOAD_FOLDER, file.filename)
        file.save(file_path)

        # Предсказание
        results = nn.predict_single_image(os.path.abspath(file_path))

        # Сохранение в БД
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute("""
            INSERT INTO images (username, name, result, probability)
            VALUES (%s, %s, %s, %s)
            RETURNING id
        """, (username, file.filename, results[0], float(results[1])))
        conn.commit()
        cursor.close()
        conn.close()

        # Формирование ответа
        diagnosis = "Меланома" if results[0] == 1 else "Не меланома"
        print(diagnosis)
```

```
        return jsonify({
            "status": "success",
            "diagnosis": diagnosis,
            "probability": float(results[1]),
            "image_path": file.filename
        }), 200

@app.route('/download/<image_name>', methods=['GET'])
def download_image(image_name):
    """Скачивание изображения"""
    file_path = os.path.join(UPLOAD_FOLDER, image_name)
    if not os.path.exists(file_path):
        return jsonify({"error": "File not found"}), 404
    return send_file(file_path, as_attachment=True)

@app.route("/login_d", methods=["POST"])
def login_doctor():
    username = request.json.get("username")
    password = request.json.get("password")
    result = get_user(username, "doctors")
    if not result or result[0][4] != password:  # Проверка пароля
        return jsonify({"error": "Invalid credentials"}), 401

    access_token = create_access_token(identity=username)
    return jsonify({
        "status": "success",
        "access_token": access_token,
        "license_number": result[0][1],
        "user_type": "doctor"
    })

@app.route("/login", methods=["POST"])
def login_patient():
    """Аутентификация пациента"""
    username = request.json.get("username")
    password = request.json.get("password")

    result = get_user(username, "patients")
    if not result or result[0][6] != password:  # Проверка пароля
        return jsonify({"error": "Invalid credentials"}), 401

    access_token = create_access_token(identity=username)
    return jsonify({
        "status": "success",
        "access_token": access_token,
        "user_type": "patient"
    })

@app.route('/patients/<license_number>', methods=['GET'])
def get_patient_list(license_number):
    try:
        result = get_patients(license_number)
        return jsonify({
            "status": "success",
            "count": len(result),
            "patients": result
        }), 200
    except Exception as e:
        app.logger.error(f"Error fetching patients: {str(e)}")
        return jsonify({"error": "Internal server error"}), 500


@app.route('/images/<email>', methods=['GET'])
```

```
def get_image_list(email):
    """Получение списка изображений пациента"""
    try:
        result = get_image_names(email)
        return jsonify({
            "status": "success",
            "count": len(result),
            "images": result
        }), 200
    except Exception as e:
        app.logger.error(f"Error fetching images: {str(e)}")
        return jsonify({"error": "Internal server error"}), 500

@app.route('/chat/send', methods=['POST'])
@jwt_required()
def send_message():
    """Отправка сообщения в чат"""
    data = request.json
    print(data)
    if not data or 'receiver' not in data or 'text' not in data:
        return jsonify({"error": "Missing required fields"}), 400

    message_id = save_chat_message(
        sender_username=data['sender'],
        receiver_username=data['receiver'],
        message_text=data['text']
    )

    if message_id:
        return jsonify({
            "status": "success",
            "message_id": message_id
        })
    return jsonify({"error": "Failed to save message"}), 500

@app.route('/chat/history', methods=['GET'])
@jwt_required()
def chat_history():
    """Получение истории сообщений"""
    receiver = request.args.get('receiver')
    sender = request.args.get('sender')
    if not receiver:
        return jsonify({"error": "Partner username is required"}), 400
    messages = get_chat_messages(sender, receiver)
    return jsonify({
        "status": "success",
        "messages": messages
    })


@app.route('/protected', methods=['GET'])
@jwt_required()
def protected():
    current_user = get_jwt_identity()
    return jsonify(logged_in_as=current_user), 200

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

ПРИЛОЖЕНИЕ Е

ПОЛНЫЙ ЛИСТИНГ ТРЁХ ACTIVITY

Исходный текст Activity, экран для отправки изображений на сервер и просмотра результата:

```
package by.overwees.mobileclient.ui

import android.app.Activity
import android.content.Context
import android.content.Intent
import android.net.Uri
import android.os.Bundle
import android.provider.MediaStore
import android.util.Log
import android.widget.Button
import android.widget.EditText
import android.widget.TextView
import androidx.activity.result.contract.ActivityResultContracts
import androidx.appcompat.app.AppCompatActivity
import androidx.appcompat.widget.AppCompatImageView
import by.overwees.mobileclient.R
import by.overwees.mobileclient.service.ImageService
import com.google.android.material.floatingactionbutton.FloatingActionButton
import java.io.ByteArrayOutputStream
import java.io.IOException
import java.io.InputStream

class MainActivity : AppCompatActivity() {

    private lateinit var pickImage: FloatingActionButton
    private lateinit var selectedImage: AppCompatImageView
    private lateinit var textViewResults: TextView
    private val baseUrl = "http://192.168.112.13:5000/"
    private lateinit var imageUploadService: ImageService
    private lateinit var btnOpenChat: Button
    private lateinit var btnLogout: Button

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        imageUploadService = ImageService(this)
        pickImage = findViewById(R.id.pick_image)
        selectedImage = findViewById(R.id.selected_image)
        textViewResults = findViewById(R.id.TextViewResult)
        btnOpenChat = findViewById(R.id.btn_open_chat)
        btnLogout = findViewById(R.id.btn_logout)
        pickImage.setOnClickListener {
            val pickImg = Intent(Intent.ACTION_PICK,
MediaStore.Images.Media.INTERNAL_CONTENT_URI)
            changeImage.launch(pickImg)
        }
        btnOpenChat.setOnClickListener {
            openChatActivity()
        }

        btnLogout.setOnClickListener {
            logout()
        }
    }
```

```
    private fun openChatActivity() {
        val sharedPrefs = getSharedPreferences("auth_prefs", MODE_PRIVATE)
        val username = sharedPrefs.getString("username", "") ?: ""

        val intent = Intent(this, ChatActivity::class.java).apply {
            putExtra("USERNAME", username)
        }
        startActivity(intent)
    }

    private fun logout() {
        getSharedPreferences("auth_prefs",
MODE_PRIVATE).edit().clear().apply()
        startActivity(Intent(this, StartActivity::class.java))
        finish()
    }
    private val changeImage =
registerForActivityResult(ActivityResultContracts.StartActivityForResult()) {
            if (it.resultCode == Activity.RESULT_OK) {
                val data = it.data
                val imgUri = data?.data
                selectedImage.setImageURI(imgUri)
                val byteArray = imgUri?.let { uri ->
uriToByteArray(this.applicationContext, uri) }
                if (byteArray != null) {
                    imageUploadService.sendImageToServer(
                        byteArray,
                        baseUrl.plus("classify")
                    ) { result ->
                        runOnUiThread {
                            if (result.success) {
                                // Успешный ответ
                                val diagnosis = result.diagnosis ?:
"Неизвестный диагноз"
                                Log.e("diag",diagnosis)
                                val probability =
result.probability?.toFloat()?.times(100)
                                textViewResults.text = "Диагноз: $diagnosis
(${"%.2f".format(probability)}%)"
                            } else {
                                textViewResults.text = result.message
                            }
                        }
                    }
                }
            }
        }

    private fun uriToByteArray(context: Context, uri: Uri): ByteArray? {
        var inputStream: InputStream? = null
        var byteArray: ByteArray? = null
        try {
            inputStream = context.contentResolver.openInputStream(uri)
            if (inputStream != null) {
                val byteArrayOutputStream = ByteArrayOutputStream()
                val buffer = ByteArray(1024)
                var length: Int
                while (inputStream.read(buffer).also { length = it } != -1) {
                    byteArrayOutputStream.write(buffer, 0, length)
                }
                byteArray = byteArrayOutputStream.toByteArray()
                byteArrayOutputStream.close()
```

```
            }
        } catch (e: IOException) {
            Log.e("UriToByteArray", "Ошибка при чтении данных из Uri", e)
        } finally {
            try {
                inputStream?.close()
            } catch (e: IOException) {
                Log.e("UriToByteArray", "Ошибка при закрытии InputStream", e)
            }
        }
        return byteArray
    }
}
```

Исходный текст Activity, содержащей экран авторизации:

```
package by.overwees.mobileclient.ui

import android.content.Intent
import android.os.Bundle
import android.widget.Button
import android.widget.EditText
import android.widget.Toast
import androidx.activity.enableEdgeToEdge
import androidx.appcompat.app.AppCompatActivity
import androidx.lifecycle.lifecycleScope
import by.overwees.mobileclient.R
import by.overwees.mobileclient.service.ApiClient
import kotlinx.coroutines.launch
class StartActivity : AppCompatActivity() {
    private val baseUrl = "http://192.168.112.13:5000/"
    private val apiClient = ApiClient(baseUrl)
    private val sharedPrefsName = "auth_prefs"
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.activity_start)
        val btnLogin: Button = findViewById(R.id.btnLogin)
        val etUsername: EditText = findViewById(R.id.editTextUsername)
        val etPassword: EditText = findViewById(R.id.editTextPassword)
        btnLogin.setOnClickListener {
            val username = etUsername.text.toString()
            val password = etPassword.text.toString()
            if (username.isBlank() || password.isBlank()) {
                showError("Заполните все поля")
                return@setOnClickListener
            }
            lifecycleScope.launch {
                try {
                    val result = apiClient.login(username, password)
                    result.onSuccess { response ->
                        if (response.access_token != null) {
                            saveToken(username, response.access_token,
response.license_number)
                            startActivity(Intent(this@StartActivity,
MainActivity::class.java))
                            finish()
                        } else {
                            showError(response.error ?: "Ошибка авторизации")
                        }
                    }.onFailure { e ->
                        showError(e.message ?: "Ошибка авторизации")
                    }
                } catch (e: Exception) {
                    showError("Ошибка сети: ${e.message}")
```

```
                }
            }
        }
    }
    private fun saveToken(username: String, token: String, licenseNumber:
String?) {
        getSharedPreferences(sharedPrefsName, MODE_PRIVATE).edit().apply {
            putString("username", username)
            putString("token", token)
            licenseNumber?.let { putString("license_number", it) }
            apply()
        }
    }
    private fun showError(message: String) {
        runOnUiThread {
            Toast.makeText(this, message, Toast.LENGTH_SHORT).show()
        }
    }
}
```

Исходный текст Activity, содержащей экран чата:

```
package by.overwees.mobileclient.ui

import android.os.Bundle
import android.widget.*
import androidx.appcompat.app.AppCompatActivity
import androidx.lifecycle.lifecycleScope
import by.overwees.mobileclient.R
import by.overwees.mobileclient.service.ApiClient
import kotlinx.coroutines.launch

class ChatActivity : AppCompatActivity() {
    private lateinit var apiClient: ApiClient
    private lateinit var currentUser: String
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_chat)
        apiClient = ApiClient("http://192.168.112.13:5000/")
        currentUser = "patient.mail@gmail.com"
        val messagesList = findViewById<ListView>(R.id.messages_list)
        val messageInput = findViewById<EditText>(R.id.message_input)
        val sendButton = findViewById<Button>(R.id.send_button)
        val closeButton = findViewById<Button>(R.id.close_button)
        sendButton.setOnClickListener {
            val message = messageInput.text.toString()
            if (message.isNotEmpty()) {
                sendMessage(message)
                messageInput.text.clear()
            }
        }
        closeButton.setOnClickListener {
            finish()
        }
        loadMessages(messagesList)
    }
    private fun loadMessages(messagesList: ListView) {
        lifecycleScope.launch {
            try {
                val sharedPrefs = getSharedPreferences("auth_prefs",
MODE_PRIVATE)
                val token = sharedPrefs.getString("token", "") ?: ""
                val response = apiClient.getChatMessages(
                    sender = 13,
                    receiver = 12,
```

```
                    token = "Bearer $token"
                )
                if (response.isSuccessful) {
                    val messages = response.body()?.messages ?: emptyList()
                    runOnUiThread {
                        val adapter = ArrayAdapter(
                            this@ChatActivity,
                            android.R.layout.simple_list_item_1,
                            messages.map { "${it.sender}: ${it.text}" }
                        )
                        messagesList.adapter = adapter
                        messagesList.setSelection(adapter.count - 1)
                    }
                } else {
                    showError("Ошибка загрузки сообщений")
                }
            } catch (e: Exception) {
                showError("Ошибка сети: ${e.message}")
            }
        }
    }
    private fun sendMessage(text: String) {
        lifecycleScope.launch {
            try {
                val sharedPrefs = getSharedPreferences("auth_prefs",
MODE_PRIVATE)
                val token = sharedPrefs.getString("token", "") ?: ""


                val response = apiClient.sendMessage(
                    sender = 13,
                    receiver = 12,
                    text = text,
                    token = "Bearer $token"
                )

                if (!response.isSuccessful) {
                    showError("Ошибка отправки сообщения")
                }
            } catch (e: Exception) {
                showError("Ошибка сети: ${e.message}")
            }
        }
    }
    private fun showError(message: String) {
        runOnUiThread {
            Toast.makeText(this@ChatActivity, message,
Toast.LENGTH_SHORT).show()
        }
    }
}
```

ПРИЛОЖЕНИЕ Ж

ПОЛНЫЙ ЛИСТИНГ СЕРВИСОВ МОБИЛЬНОГО КЛИЕНТА

Исходный текст сервиса авторизации:

```
package by.overwees.mobileclient.service

import android.util.Log
import by.overwees.mobileclient.R
import okhttp3.OkHttpClient
import retrofit2.Response
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory
import retrofit2.http.Body
import retrofit2.http.GET
import retrofit2.http.Header
import retrofit2.http.POST
import retrofit2.http.Query
import java.math.BigInteger
import java.security.MessageDigest

interface AuthApiService {
    @POST("login")
    suspend fun login(@Body request: LoginRequest): LoginResponse

    @GET("chat/history")
    suspend fun getChatMessages(
        @Query("sender") sender: Int,
        @Query("receiver") receiver: Int,
        @Header("Authorization") token: String
    ): Response<ChatMessagesResponse>

    @POST("chat/send")
    suspend fun sendMessage(
        @Body message: SendMessageRequest,
        @Header("Authorization") token: String
    ): Response<BasicResponse>
}

data class LoginRequest(val username: String, val password: String)
data class LoginResponse(
    val access_token: String?,
    val error: String?,
    val license_number: String?
)

data class ChatMessage(
    val sender: Int,
    val text: String,
    val timestamp: String
)

data class ChatMessagesResponse(
    val messages: List<ChatMessage>
)

data class SendMessageRequest(
    val sender: Int,
    val receiver: Int,
```

```
    val text: String
)

data class BasicResponse(
    val success: Boolean
)

class ApiClient(baseUrl: String) {
    private val retrofit = Retrofit.Builder()
        .baseUrl(baseUrl)
        .client(OkHttpClient.Builder().build())
        .addConverterFactory(GsonConverterFactory.create())
        .build()

    private val authService = retrofit.create(AuthApiService::class.java)

    suspend fun login(username: String, password: String):
Result<LoginResponse> {
        return try {
            val response = authService.login(LoginRequest(md5Hash(username),
md5Hash(password)))
            if (response.access_token != null) {
                Result.success(response)
            } else {
                Result.failure(Exception(response.error ?: "Unknown error"))
            }
        } catch (e: Exception) {
            Result.failure(e)
        }
    }
    suspend fun getChatMessages(
        sender: Int,
        receiver: Int,
        token: String
    ): Response<ChatMessagesResponse> {
        return authService.getChatMessages(sender, receiver, token)
    }
    suspend fun sendMessage(
        sender: Int,
        receiver: Int,
        text: String,
        token: String
    ): Response<BasicResponse> {
        return authService.sendMessage(
            SendMessageRequest(sender, receiver, text),
            token
        )
    }
}

fun md5Hash(input: String): String {
    val md = MessageDigest.getInstance("MD5")
    val digest = md.digest(input.toByteArray(Charsets.UTF_8))
    return BigInteger(1, digest).toString(16).padStart(32, '0')
}
```

Исходный текст сервиса отправки изображений:

```
package by.overwees.mobileclient.service

import android.content.Context
import android.graphics.Bitmap
import android.graphics.BitmapFactory
import okhttp3.*
import okhttp3.MediaType.Companion.toMediaTypeOrNull
```

```
import okhttp3.RequestBody.Companion.asRequestBody
import org.json.JSONObject
import java.io.File
import java.io.FileOutputStream
import java.io.IOException
import java.text.SimpleDateFormat
import java.util.*

class ImageService(private val context: Context) {
    data class DiagnosisResult(
        val success: Boolean,
        val message: String? = null,
        val diagnosis: String? = null,
        val probability: Double? = null
    )
    fun sendImageToServer(
        byteArray: ByteArray,
        serverUrl: String,
        callback: (DiagnosisResult) -> Unit
    ) {
        val bitmap: Bitmap? = BitmapFactory.decodeByteArray(byteArray, 0,
byteArray.size)
        if (bitmap == null) {
            callback(DiagnosisResult(false, "Ошибка: Невозможно преобразовать
ByteArray в Bitmap"))
            return
        }
        val sharedPrefs = context.getSharedPreferences("auth_prefs",
Context.MODE_PRIVATE)
        val username = sharedPrefs.getString("username", "unknown_user") ?:
"unknown_user"
        val token = sharedPrefs.getString("token", null)

        if (token.isNullOrEmpty()) {
            callback(DiagnosisResult(false, "Ошибка: Токен авторизации не
найден"))
            return
        }
        val timeStamp = SimpleDateFormat("yyyyMMdd_HHmmss",
Locale.getDefault()).format(Date())
        val fileName = "${username}_$timeStamp"
        val outputDir = File.createTempFile("temp_dir", "").parentFile
        val outputFile = File(outputDir, "$fileName.jpg")
        val isSaved = saveBitmapToJpg(bitmap, outputFile)
        if (!isSaved) {
            callback(DiagnosisResult(false, "Ошибка: Невозможно сохранить
Bitmap в файл"))
            return
        }
        uploadFileToServer(outputFile, serverUrl, token, callback)
    }
    private fun saveBitmapToJpg(bitmap: Bitmap, file: File): Boolean {
        return try {
            val outputStream = FileOutputStream(file)
            bitmap.compress(Bitmap.CompressFormat.JPEG, 100, outputStream)
            outputStream.flush()
            outputStream.close()
            true
        } catch (e: Exception) {
            e.printStackTrace()
            false
        }
    }
```

```
    private fun uploadFileToServer(
        file: File,
        serverUrl: String,
        authToken: String,
        callback: (DiagnosisResult) -> Unit
    ) {
        val client = OkHttpClient()
        val requestBody: RequestBody = MultipartBody.Builder()
            .setType(MultipartBody.FORM)
            .addFormDataPart(
                "file",
                file.name,
                file.asRequestBody("image/jpeg".toMediaTypeOrNull())
            )
            .build()
        val request = Request.Builder()
            .url(serverUrl)
            .header("Authorization", "Bearer $authToken")
            .post(requestBody)
            .build()
        client.newCall(request).enqueue(object : Callback {
            override fun onFailure(call: Call, e: IOException) {
                callback(DiagnosisResult(false, "Ошибка сети: ${e.message}"))
            }
            override fun onResponse(call: Call, response: Response) {
                try {
                    if (response.isSuccessful) {
                        val responseBody = response.body?.string()
                        val json = JSONObject(responseBody ?: "{}")
                        val diagnosis = json.optString("diagosis")
                        val probability = json.optDouble("probability")
                        callback(DiagnosisResult(
                            success = true,
                            message = json.optString("message"),
                            diagnosis = diagnosis,
                            probability = probability
                        ))
                    } else {
                        when (response.code) {
                            401 -> callback(DiagnosisResult(false, "Ошибка
авторизации"))
                            403 -> callback(DiagnosisResult(false, "Доступ
запрещен"))
                            else -> callback(DiagnosisResult(false,
                                "Ошибка сервера: ${response.code} -
${response.message}"))
                        }
                    }
                } catch (e: Exception) {
                    callback(DiagnosisResult(false, "Ошибка обработки ответа:
${e.message}"))
                }
            }
        })
    }
}
```

ПРИЛОЖЕНИЕ И

ПОЛНЫЙ ЛИСТИНГ КЛАССА ИНИЦИАЛИЗАЦИИ ВИЗУАЛЬНОГО ИНТЕРФЕЙСА

```
from PIL import Image , ImageTk
import PIL
from tkinter import *
import tkinter as tk
from tkinter import messagebox
from tkinter import ttk
from serviceAPI import APIClient
#
class App:

    def __init__(self, root):
        self.root = root
        self.root.title("Doctor's client")
        self.root.resizable(0, 0)
        self.api = APIClient()  # Наш клиент для работы с API
        self.selected_patient_FIO = ""
        self.image_list = []
        self.current_image_id = 0
        self.image_description = StringVar(value="Имя файла:\n диагноз:\n
вероятность:")
        # Создаем контейнеры для разных экранов
        self.login_frame = tk.Frame(self.root, padx=10, pady=10)
        self.main_frame = tk.Frame(self.root, padx=10, pady=10)

        self.create_login_window()
        self.create_main_window()

        # Показываем начальный экран
        self.show_login_screen()

    def create_main_window(self):
        """Создание главного интерфейса"""
        # Основные элементы главного экрана
        self.create_table()

        # Кнопки управления
        button_frame = Frame(self.main_frame)
        button_frame.grid(row=4, column=0, columnspan=4, pady=10)

        self.chat_btn = Button(
            button_frame,
            text="Открыть чат",
            command=self.open_chat_window,
            state=DISABLED
        )
        self.chat_btn.pack(side=LEFT, padx=5)

        self.logout_btn = Button(
            button_frame,
            text="Выйти",
            command=self.handle_logout
        )
        self.logout_btn.pack(side=LEFT, padx=5)

        self.create_image_gallery()
```

```
    def create_table(self):
        self.table = ttk.Treeview(self.main_frame)
        # Определяем колонки
        self.table['columns'] = ('FIO', 'Birth date', 'Gender', 'Phone',
'Email')

        # Форматируем колонки
        self.table.column("#0", width=0, stretch=tk.NO)  # Скрытая колонка
        self.table.column("FIO", anchor=tk.CENTER, width=200)
        self.table.column("Birth date", anchor=tk.W, width=70)
        self.table.column("Gender", anchor=tk.W, width=30)
        self.table.column("Phone", anchor=tk.W, width=100)
        self.table.column("Email", anchor=tk.W, width=160)

        # Создаем заголовки
        self.table.heading("#0", text="", anchor=tk.W)
        self.table.heading("FIO", text="ФИО", anchor=tk.CENTER)
        self.table.heading("Birth date", text="Дата рождения", anchor=tk.W)
        self.table.heading("Gender", text="Пол", anchor=tk.W)
        self.table.heading("Phone", text="Телефон", anchor=tk.W)
        self.table.heading("Email", text="Email", anchor=tk.W)

        # Добавляем полосу прокрутки
        scroll = ttk.Scrollbar(self.main_frame, orient=tk.VERTICAL,
command=self.table.yview)
        self.table.configure(yscroll=scroll.set)
        # Привязываем обработчик клика
        self.table.bind('<ButtonRelease-1>', self.on_table_row_click)  #
Левая кнопка мыши
        scroll.grid(row=0, column=4, rowspan=4, columnspan=1,sticky="ns")
        self.table.grid(row=0, column=0, rowspan=4, columnspan=4,sticky="ns")

    def create_login_window(self):
        """Создание интерфейса входа"""
        # Поля ввода
        tk.Label(self.login_frame, text="Логин:").grid(row=0, column=0,
padx=10, pady=5, sticky="e")
        self.username_entry = tk.Entry(self.login_frame)
        self.username_entry.grid(row=0, column=1, padx=10, pady=5)

        tk.Label(self.login_frame, text="Пароль:").grid(row=1, column=0,
padx=10, pady=5, sticky="e")
        self.password_entry = tk.Entry(self.login_frame, show="*")
        self.password_entry.grid(row=1, column=1, padx=10, pady=5)

        # Кнопки
        self.login_btn = tk.Button(self.login_frame, text="Вход",
command=self.handle_login)
        self.login_btn.grid(row=2, column=0, columnspan=2, pady=10)

        # Тестовые данные для удобства
        self.username_entry.insert(0, "doctor_login")
        self.password_entry.insert(0, "DoctorPassword123")

    def show_login_screen(self):
        """Показать экран входа"""
        self.main_frame.pack_forget()  # Скрыть главный экран
        self.login_frame.pack()  # Показать экран входа

    def show_main_screen(self):
        """Показать главный экран"""
        self.login_frame.pack_forget()  # Скрыть экран входа
        self.main_frame.pack()  # Показать главный экран
```

```
    def handle_login(self):
        """Обработка входа"""
        username = self.username_entry.get()
        password = self.password_entry.get()

        if not username or not password:
            messagebox.showerror("Error", "Username and password are
required")
            return

        response = self.api.login_doctor(username, password)

        if response["success"]:
            self.handle_table_loading()
            self.show_main_screen()
        else:
            error_msg = response.get("error", response["data"].get("msg",
"Login failed"))
            messagebox.showerror("Error", error_msg)

    def handle_table_loading(self):
        # Добавляем данные
        response = self.api.get_patients_by_license(self.api.license_number)
        for row in self.table.get_children():
            self.table.delete(row)
        for record in response[1]["patients"]:
            self.table.insert(parent='', index='end', values=record)

    def handle_logout(self):
        """Обработка выхода из системы"""
        self.api.logout()
        self.show_login_screen()

    def on_table_row_click(self, event):
        """Обработка выбора строки в таблице"""
        selected_item = self.table.focus()
        if not selected_item:
            return

        item_data = self.table.item(selected_item)
        values = item_data['values']
        self.selected_patient_email = values[-1]  # сохраняем email пациента
        self.selected_patient_FIO = values[0]
        self.chat_btn.config(state=NORMAL)
        response = self.api.get_image_list(self.selected_patient_email)
        self.image_list = []
        self.current_image_id = 0

        if response[0]:  # если успешно
            for i in range(response[1]['count']):
                self.image_list.append(response[1]['images'][i])
            self.show_current_image()
        self.table.tag_configure('selected', background='#e0e0ff')
        self.table.item(selected_item, tags=('selected',))

    def create_image_gallery(self):
        self.current_image_label = ttk.Label(self.main_frame)
        self.current_image_label.grid(row=0, column=5, rowspan=2,
columnspan=2)

        # Описание снимка
        self.image_desc_label = ttk.Label(
```

```
            self.main_frame,
            wraplength=400,
            font=('Arial', 10),
            justify=tk.CENTER,
            textvariable= self.image_description
        )
        self.image_desc_label.grid(row=2, column=5, rowspan=1, columnspan=2)

        # Навигация
        nav_frame = ttk.Frame(self.main_frame)
        nav_frame.grid(row=3, column=5, rowspan=1, columnspan=2)

        self.prev_btn = ttk.Button(
            nav_frame,
            text="<< Предыдущий",
            command=self.show_prev_image
        )
        self.prev_btn.pack(side=tk.LEFT, padx=5)

        self.next_btn = ttk.Button(
            nav_frame,
            text="Следующий >>",
            command=self.show_next_image
        )
        self.next_btn.pack(side=tk.LEFT, padx=5)

    def show_current_image(self):

        # Загружаем изображение с сервера
        try:
            response =
self.api.get_image(self.image_list[self.current_image_id][2])
            # Конвертируем в формат для Tkinter
            pil_image = PIL.Image.open(response[1])
            # Масштабируем (максимальный размер 500x500)
            pil_image.thumbnail((500, 500))
            tk_image = ImageTk.PhotoImage(pil_image)
            # Сохраняем ссылку, чтобы изображение не удалилось
            self.current_image_label.tk_image = tk_image
            self.current_image_label.config(image=tk_image, text="")
            tmp_description=f"Имя файла:
{self.image_list[self.current_image_id][2]}\n Диагноз:"

            if self.image_list[self.current_image_id][3] == 1:
                tmp_description += "Меланома\n"
            else:
                tmp_description += "Отсутствие меланомы\n"
                tmp_description += f"Вероятность:
{self.image_list[self.current_image_id][-1] * 100:.2f}%"

            self.image_description.set(tmp_description)
        except Exception as e:
            self.current_image_label.config(image='', text=f"Ошибка загрузки:
{str(e)}")
            print(str(e))

    def show_next_image(self):
        if self.current_image_id < len(self.image_list) - 1:
            self.current_image_id += 1
        else:
            self.current_image_id = 0
        self.show_current_image()
```

```
    def show_prev_image(self):
        if self.current_image_id > 0:
            self.current_image_id -= 1
        else:
            self.current_image_id = len(self.image_list) - 1
        self.show_current_image()

    def open_chat_window(self):
        """Открытие окна чата"""
        if not self.selected_patient_email:
            return
        chat_window = Toplevel(self.root)
        chat_window.title(f"Чат с {self.selected_patient_email}")

        # Элементы чата
        messages_frame = Frame(chat_window)
        scrollbar = Scrollbar(messages_frame)
        messages_list = Listbox(messages_frame, yscrollcommand=scrollbar.set)
        scrollbar.config(command=messages_list.yview)

        message_entry = Entry(chat_window, width=50)
        send_btn = Button(chat_window, text="Отправить",
                          command=lambda: self.send_chat_message(
                              message_entry.get(),
                              messages_list,
                              self.selected_patient_email
                          ))
        messages_frame.pack(fill=BOTH, expand=True)
        scrollbar.pack(side=RIGHT, fill=Y)
        messages_list.pack(fill=BOTH, expand=True)
        message_entry.pack(side=LEFT, fill=X, expand=True)
        send_btn.pack(side=RIGHT)

        # Загрузка истории
        response = self.api.get_chat_messages(
            self.api.license_number,
            self.selected_patient_email
        )

        if response["success"]:
            for msg in response["data"].get("messages", []):
                prefix = "Вы: " if msg["sender_id"] == 12 else "Пациент: "
                messages_list.insert(END, f"{prefix}{msg['text']}")

    def send_chat_message(self, message: str, messages_list: Listbox,
receiver: str):
        """Отправка сообщения"""
        if not message:
            return

        response = self.api.send_chat_message(
            self.api.license_number,
            receiver,
            message
        )

        if response["success"]:
            messages_list.insert(END, f"Вы: {message}")
        else:
            messagebox.showerror("Ошибка", "Не удалось отправить сообщение")
```

ПРИЛОЖЕНИЕ К

ПОЛНЫЙ ЛИСТИНГ КЛАССА СЕРВИСА

```
from io import BytesIO
import requests
import json
from typing import Optional, Dict, Any, Tuple
import hashlib
from PIL.Image import Image
from requests import RequestException


def md5_encode(x: str) -> str:
    """Хеширование строки по MD5"""
    return hashlib.md5(x.encode('utf-8')).hexdigest()


class APIClient:
    def __init__(self, base_url: str = "http://192.168.112.13:5000"):
        self.base_url = base_url
        self.license_number = ""
        self.token: Optional[str] = None
        self.session = requests.Session()  # Используем сессию для сохранения
кук

    def login_doctor(self, username: str, password: str) -> Dict[str, Any]:
        """Аутентификация врача через /login_d"""
        url = f"{self.base_url}/login_d"
        payload = {
            "username": md5_encode(username),
            "password": md5_encode(password)
        }

        try:
            response = self.session.post(url, json=payload)
            data = response.json()

            if response.status_code == 200:
                self.token = data.get('access_token')
                self.license_number = data.get("license_number", "")
                print(f"Успешная авторизация. Токен: {self.token}")

            return {
                "success": response.status_code == 200,
                "status_code": response.status_code,
                "data": data
            }
        except RequestException as e:
            return {
                "success": False,
                "error": str(e),
                "status_code": None
            }

    def get_patients_by_license(self, license_number: str) -> Tuple[bool,
Any]:
        """Получение пациентов врача через /patients/<license>"""
        url = f"{self.base_url}/patients/{license_number}"
        try:
            response = self.session.get(url)
            response.raise_for_status()
```

```
            return True, response.json()
        except RequestException as e:
            error_msg = f"Ошибка запроса: {str(e)}"
            if response := getattr(e, 'response', None):
                try:
                    error_details = response.json()
                    if 'error' in error_details:
                        error_msg += f" | {error_details['error']}"
                except:
                    pass
            return False, error_msg

    def get_image_list(self, email: str) -> Tuple[bool, Any]:
        """Получение списка изображений через /images/<email>"""
        url = f"{self.base_url}/images/{email}"
        try:
            response = self.session.get(url)
            response.raise_for_status()
            return True, response.json()
        except RequestException as e:
            return False, f"Ошибка запроса: {str(e)}"

    def get_image(self, image_name: str) -> Tuple[bool, BytesIO]:
        """Загрузка изображения через /download/<image_name>"""
        url = f"{self.base_url}/download/{image_name}"
        try:
            response = self.session.get(url)
            response.raise_for_status()
            return True, BytesIO(response.content)
        except RequestException as e:
            return False, BytesIO()

    def get_chat_messages(self, sender: str, receiver: str) -> Dict[str,
Any]:
        """Получение истории чата через /chat/history"""
        if not self.token:
            return {"success": False, "error": "Токен отсутствует",
"status_code": 401}

        url = f"{self.base_url}/chat/history"
        headers = {"Authorization": f"Bearer {self.token}"}
        params = {"sender":12, "receiver": 13}

        try:
            response = self.session.get(url, headers=headers, params=params)
            return {
                "success": response.status_code == 200,
                "status_code": response.status_code,
                "data": response.json()
            }
        except RequestException as e:
            return {
                "success": False,
                "error": str(e),
                "status_code": None
            }

    def send_chat_message(self, sender: str, receiver: str, message: str) ->
Dict[str, Any]:
        """Отправка сообщения через /chat/send"""
        if not self.token:
            return {"success": False, "error": "Токен отсутствует",
"status_code": 401}
```

```
        url = f"{self.base_url}/chat/send"
        headers = {
            "Authorization": f"Bearer {self.token}",
            "Content-Type": "application/json"
        }
        payload = {
            "sender": 12,
            "receiver": 13,
            "text": message
        }

        try:
            response = self.session.post(url, json=payload, headers=headers)
            return {
                "success": response.status_code == 200,
                "status_code": response.status_code,
                "data": response.json()
            }
        except RequestException as e:
            return {
                "success": False,
                "error": str(e),
                "status_code": None
            }

    def logout(self):
        """Очистка данных сессии"""
        self.token = None
        self.license_number = ""
        self.session = requests.Session()
```