МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

ANDROID-ПРИЛОЖЕНИЕ ДЛЯ КЛАССИФИКАЦИИ ИЗОБРАЖЕНИЙ

Студент 30.05.25 (подпись, дата) Костюков Илья Дмитриевич (ФИО полностью)

Руководитель 30.05.25 (подпись, дата) Сорокин А.А. (фамилия, инициалы)

Нормоконтролёр 30.05.25 (подпись, дата) Бычков А.С. (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 28.05.2025 г. № 16)

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

Работа защищена с отметкой Отлично (протокол заседания ГЭК от 2 июня 2025 г. № 5)

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

Пенза 2025

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

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
_____М.А. Митрохин
«__6__»__мая___ 2025___г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Костюков Илья Дмитриевич гр. 21ВВП2 факультета ВТ

2.Тема работы Android-приложение для классификации изображений

Тема утверждена приказом ПГУ №601/о от "6" мая 2025 г.

3. Руководитель бакалаврской работы Сорокин А.А.

4. Задание на работу (назначение разработки, исходные данные и т.п.)
Разработать мобильное приложение для устройств на базе ОС Android для локальной классификации изображений с помощью встроенной модели свёрточной нейронной сети.

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметной области: изучить существующие реализации и подходы. Выбрать язык программирования, среду разработки, СУБД, модель нейронной сети. Разработать архитектуру приложения и реализовать заданный функционал.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	24	06.05.2025-08.05.2025	
2	Изучение аналогов	16	09.05.2025-10.05.2025	
3	Анализ требований к приложению	16	11.05.2025-12.05.2025	
4	Выбор технологий разработки	24	13.05.2025-15.05.2025	
5	Обучение модели	32	16.05.2025-19.05.2025	
6	Разработка приложения	48	20.05.2025-25.05.2025	
7	Тестирование приложения	16	26.05.2025-27.05.2025	
8	Оформление пояснительной записки	24	28.05.2025-30.05.2025	

Дата выдачи задания "__6__"__мая__2025 г.

Руководитель бакалаврской работы _____ 06.05.25 _____ Сорокин А.А.
(подпись, дата)

Задание к исполнению принял студент _____ 06.05.25 _____ Костюков И.Д.
(подпись, дата)

Работу к защите допустить
Декан факультета _____ 30.05.2025 _____ Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 88 листов, 24 рисунка, 2 таблицы, 6 источников, 3 приложения.

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЙ, СВЁРТОЧНАЯ НЕЙРОННАЯ СЕТЬ, ТРАНСФЕРНОЕ ОБУЧЕНИЕ, TENSORFLOW LITE, MVVM, SQLITE, ANDROID STUDIO, MOBILENETV2

Цель работы – разработать Android-приложение для локальной классификации изображений с использованием свёрточной нейронной сети.

Одним из ключевых решений для реализации поставленной задачи стало создание мобильного приложения, обеспечивающего классификацию изображений по шести категориям («Авто», «Документы», «Еда», «Животные», «Люди», «Не определено») с использованием свёрточной нейронной сети, а также взаимодействие с системной галереей устройства для доступа ко всем фотографиям пользователя и интуитивно-понятный интерфейс для просмотра результатов в виде папок.

В результате разработано мобильное приложение, осуществляющее полностью автономную классификацию всех пользовательских изображений. Для достижения результата применены технологии: java, python, SQLite, MVVM, TensorFlow Lite.

Содержание

Перечень сокращений и обозначений 5
Введение 6
1 Android-приложение для классификации изображений 7
1.1 Анализ темы 7
1.2 Обзор аналогов 8
1.3 Постановка задачи 12
1.4 Архитектура приложения 13
1.5 Выбор языка программирования 15
1.6 Выбор СУБД 16
1.7 Выбор CNN-модели 16
1.8 Выбор среды разработки 17
2 Разработка приложения 19
2.1 Обучение модели 19
2.1.1 Создание набора данных 20
2.1.2 Создание модели 20
2.1.3 Обучение модели 22
2.1.4 Тонкая настройка 24
2.1.5 Оценка и экспорт модели 25
2.2 Разработка компонентов приложения 28
2.2.1 Разработка Model 28
2.2.2 Разработка View 34
2.2.3 Разработка ViewModel 35
3 Тестирование приложения 38
4 Безопасность жизнедеятельности 48
4.1 Правила безопасности при использовании смартфона 48
Заключение 50
Список использованных источников 51
Приложение А – UML-диаграммы 52

Приложение Б – Листинг программы 56
Приложение В – Презентация 81

Перечень сокращений и обозначений

БД – база данных

ИИ – искусственный интеллект

ОС – операционная система

СУБД – система управления базами данных

Точность Top-1 – доля правильно классифицированных изображений, когда модель делает единственный лучший выбор

Точность Top-5 – доля правильно классифицированных изображений, когда правильная метка находится в списке из пяти наиболее вероятных предположений модели

ID (Identifier) – идентификатор

IDE (Integrated Development Environment) – интегрированная среда разработки

MVVM (Model-View-ViewModel) – шаблон проектирования архитектуры приложения

UI (User Interface) – пользовательский интерфейс

UML (Unified Modeling Language) – унифицированный язык моделирования

URI (Uniform Resource Identifier) – унифицированный идентификатор ресурса

Введение

Современный этап развития информационных технологий характеризуется стремительным ростом внедрения искусственного интеллекта в жизнь человека, причём ключевую роль в этом процессе играют мобильные устройства. Смартфоны на базе Android, которые с каждым годом получают более мощные процессоры, в некоторых случаях даже графические ускорители, становятся приоритетными платформами для интеграции в них нейронных сетей. Одним из наиболее востребованных направлений в этой области является компьютерное зрение, а именно классификация изображений, которая находит применение в самых разных сферах – от автоматической сортировки фотографий до биометрической аутентификации по лицу.

Возможность выполнения таких задач непосредственно на мобильных устройствах стала реальной благодаря развитию мобильно-оптимизированных нейронных сетей, которые сочетают в себе высокую точность с низким энергопотреблением, что позволяет обрабатывать данные локально, не отправляя данные за пределы телефона.

Актуальность данной темы обусловлена не только широким распространением мобильных телефонов на базе операционной системы Android и ростом вычислительных мощностей смартфонов, но и растущим спросом на приложения, которые могут обрабатывать большое количество визуальной информации в реальном времени.

Цель данной работы – разработка Android-приложения с применением встроенной модели свёрточной нейронной сети для классификации изображений локально на устройстве.

1 Android-приложение для классификации изображений

1.1 Анализ темы

Классификация изображений используется во многих сферах нашей жизни. Например, в здравоохранении медицинские изображения могут автоматически анализироваться с использованием классификации изображений. Это позволяет медицинскому персоналу более точно и эффективно диагностировать и лечить пациентов. Не менее важную роль классификация изображений играет в системах машинного зрения для автономных и беспилотных автомобилей. С её помощью автомобиль может корректно реагировать на изменяющиеся условия, различать сигналы светофоров и разрешающие или запрещающие знаки. Кроме того, высокоскоростная обработка изображений позволяет транспортным средствам принимать решения в реальном времени, что критически важно для обеспечения безопасности на дороге.

Особое внимание нужно уделить сфере безопасности, в которой классификация изображений может быть использована следующими способами:

– автоматическая идентификация и классификации объектов на кадрах видеонаблюдения может помочь сотрудникам службы безопасности быстрее обнаруживать потенциальные угрозы и реагировать на них;

– распознавание лиц может помочь властям идентифицировать людей, представляющих интерес, в толпе. Классификация изображений может помочь сектору безопасности использовать технологии компьютерного зрения в таких областях, как банки, аэропорты и другие места массового скопления людей.

Также классификация изображений используется для улучшения рекомендаций пользователей в социальных сетях на основе их интересов, что позволяет персонализировать ленту новостей. При модерировании контента

эта технология помогает обнаруживать изображения, которые нарушают правила платформы.

Все вышеперечисленные примеры описывают применение классификации изображений на глобальном уровне. Но она может быть полезна и обычным людям. У каждого из нас в мобильном устройстве есть огромное количество фотографий. Найти нужную фотографию среди такого множества – иногда очень большая проблема. Именно использование классификации изображений позволяет намного упростить данную задачу.

1.2 Обзор аналогов

Google Фото [1] – сервис, представленный компаний Google, для хранения, редактирования, обмена фотографиями и видео. Помимо данных функций он предоставляет большие возможности по поиску и сортировки фотографий пользователя. Контент сортируется по разным категориям: место съёмки, лица и объекты, изображённые на фото или видео. На рисунке 1 можно увидеть отсортированные изображения документов по разным категориям.

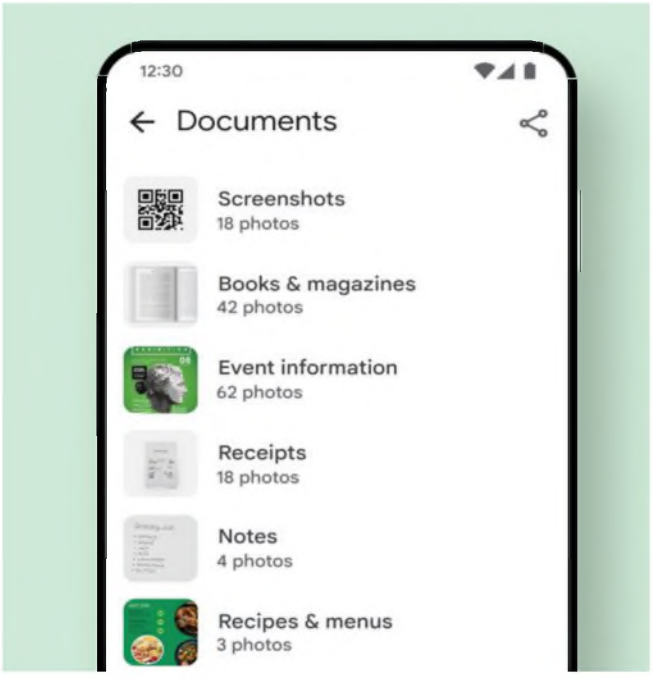


Рисунок 1 – Классификация документов

Из недостатков данного приложения можно выделить:

– для работы требуется доступ в интернет;

– ограниченное хранилище данных (15 ГБ). После заполнения требуется подписка.

Google Объектив [2] – это разработанная компаний Google технология распознавания изображений, используемая для получения информации об объектах, полученных с камеры устройства, которые она распознаёт с помощью анализа нейронной сети. Рисунок 2 демонстрирует, как при наведении камеры телефона на объект Google Объектив идентифицирует объект.

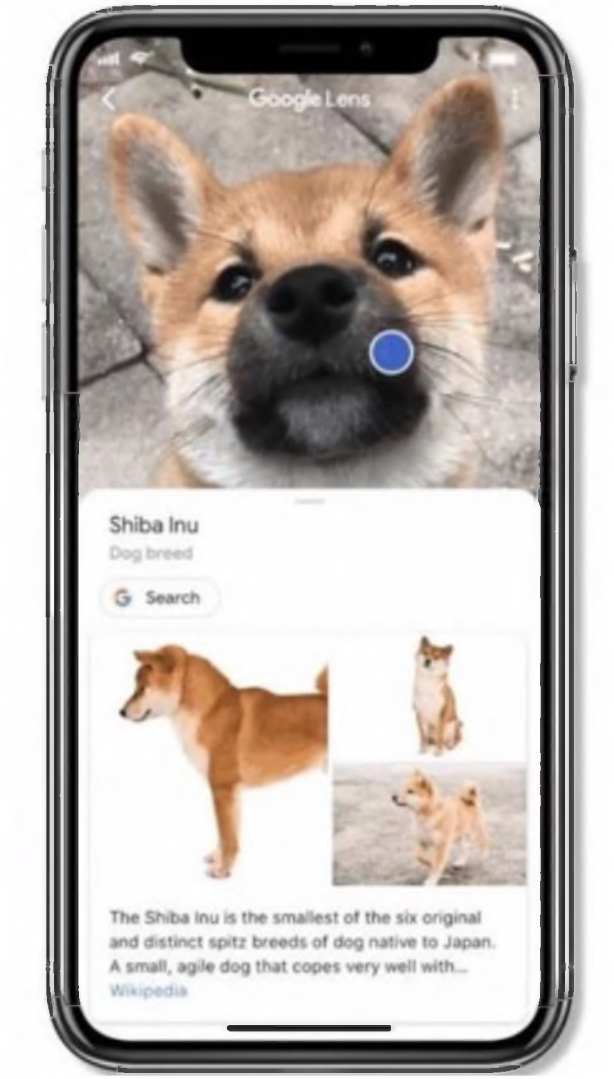


Рисунок 2 – Идентификация объекта

Недостатки данного приложения заключаются в следующем:

– для распознавания содержимого нужен интернет;

– классификация происходит только для одного изображения.

Галерея ИИ (AI Gallery) [3] – приложение, созданное командой KX Camera Team, схожее по функционалу с Google Фото. Главное отличие от Google Фото и в тоже время достоинство данного продукта – для сортировки фотографий пользователя не нужен доступ в интернет. Пример сортировки показан на рисунке 3.

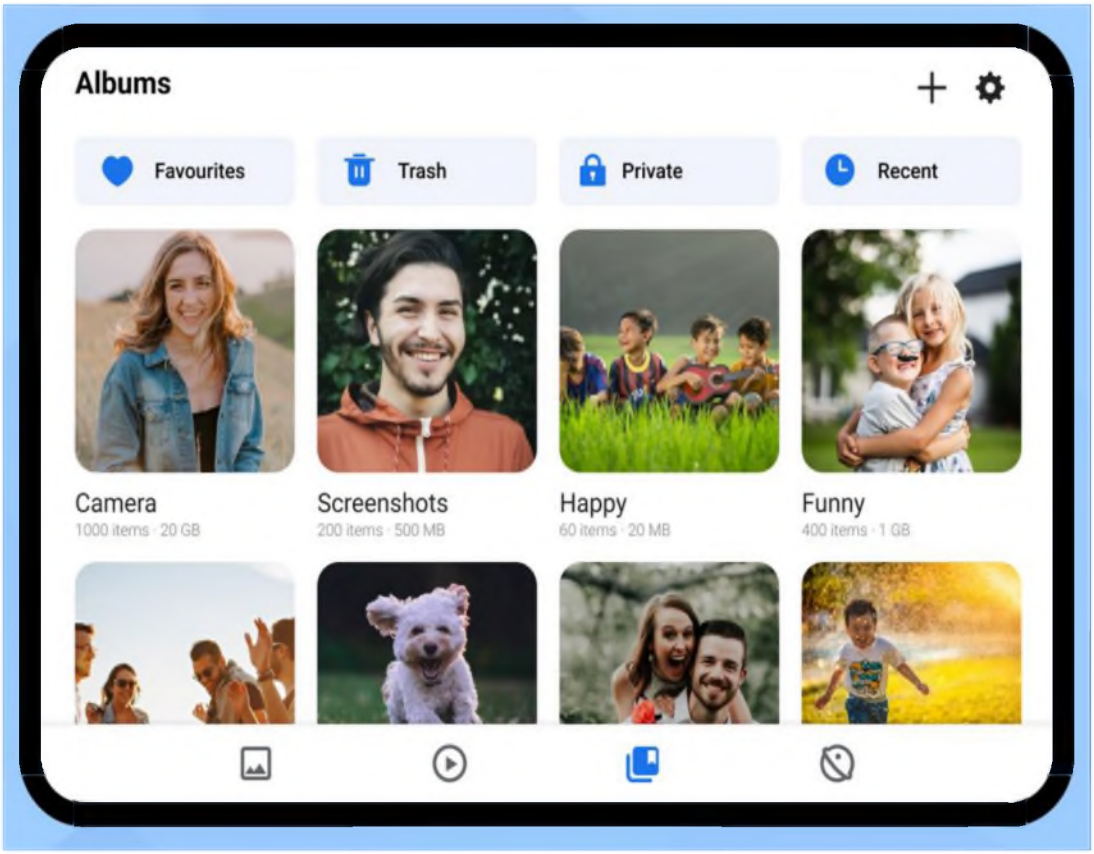


Рисунок 3 – Сортировка изображений

Недостатками данного приложения являются навязчивая реклама в бесплатной версии и платная подписка для получения некоторых функций.

Smart Image Classifier [4] – мобильное приложение, предназначенное для предоставления возможности легко классифицировать изображения без необходимости подключения к Интернету. Преимущество данного

приложения заключается в том, что оно работает автономно. У данного приложения есть два режима работы:

– классификация изображения с камеры телефона: обработка изображения, которое получено с камеры;

– классификация изображения из галереи телефона: пользователь сам выбирает изображение, которое нужно обработать.

На рисунке 4 показано главное меню приложения.

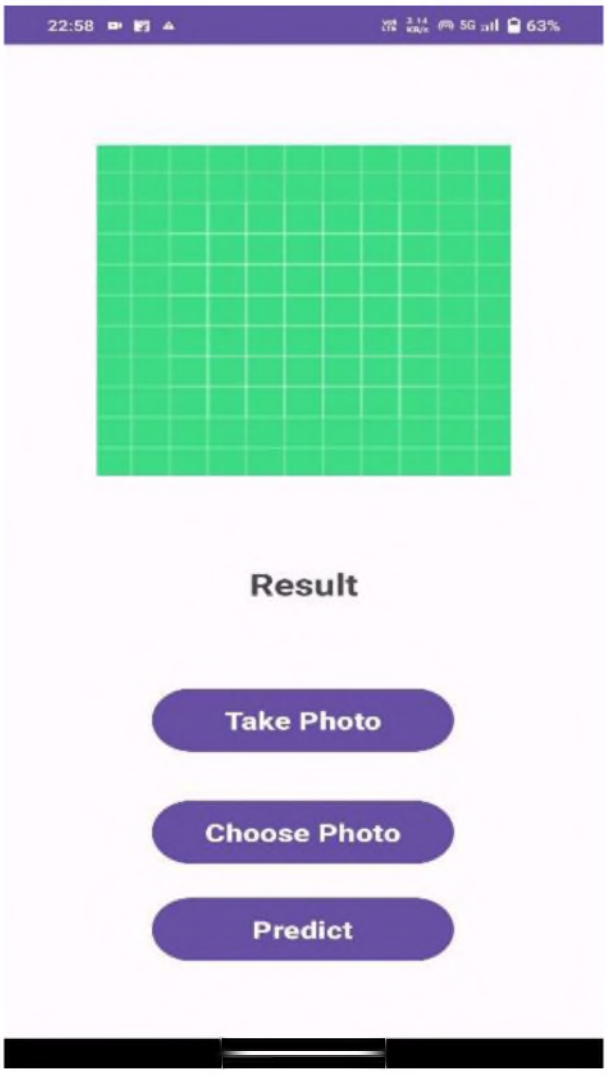


Рисунок 4 – Главное меню приложения

Главный недостаток данного приложения – с помощью него нельзя обработать сразу большое количество изображений.

В таблице 1 приведено сравнение аналогов между собой.

Таблица 1 – Сравнение аналогов

	Критерии		
Название продукта	Доступ в Интернет	Обработка нескольких изображений	Цена
Google Фото	Да	Да	Freemium
Google Объектив	Да	Нет	Бесплатно
Галерея ИИ	Нет	Да	Freemium
Smart Image Classifier	Нет	Нет	Бесплатно

1.3 Постановка задачи

Исходя из вышеперечисленных примеров, функциональные требования приложения для классификации изображений должны быть следующими:

– приложение должно быть полностью автономным, т.е. предполагается его использование без доступа к Интернету. Все вычисления будут производиться на устройстве;

– приложение будет обрабатывать все фотографии, хранящиеся на телефоне;

– пользователю будет дана возможность просмотра итогового результата. В данном случае – просмотр содержимого папок разных категорий, в которых находятся изображения, попадающие под данный класс. Всего будет предоставлено 6 категорий: «Авто», «Документы», «Еда», «Животные», «Люди», «Не определено»;

– приложение должно иметь интуитивно понятный интерфейс.

Минимальные требования к аппаратной части:

– версия ОС Android должна быть больше или равна 8.0;

– процессор на базе ARM 64;

– наличие более 2 ГБ оперативной памяти для работы приложения;

– наличие более 50 МБ свободного пространства.

UML-диаграмма вариантов использования представлена на рисунке 5.

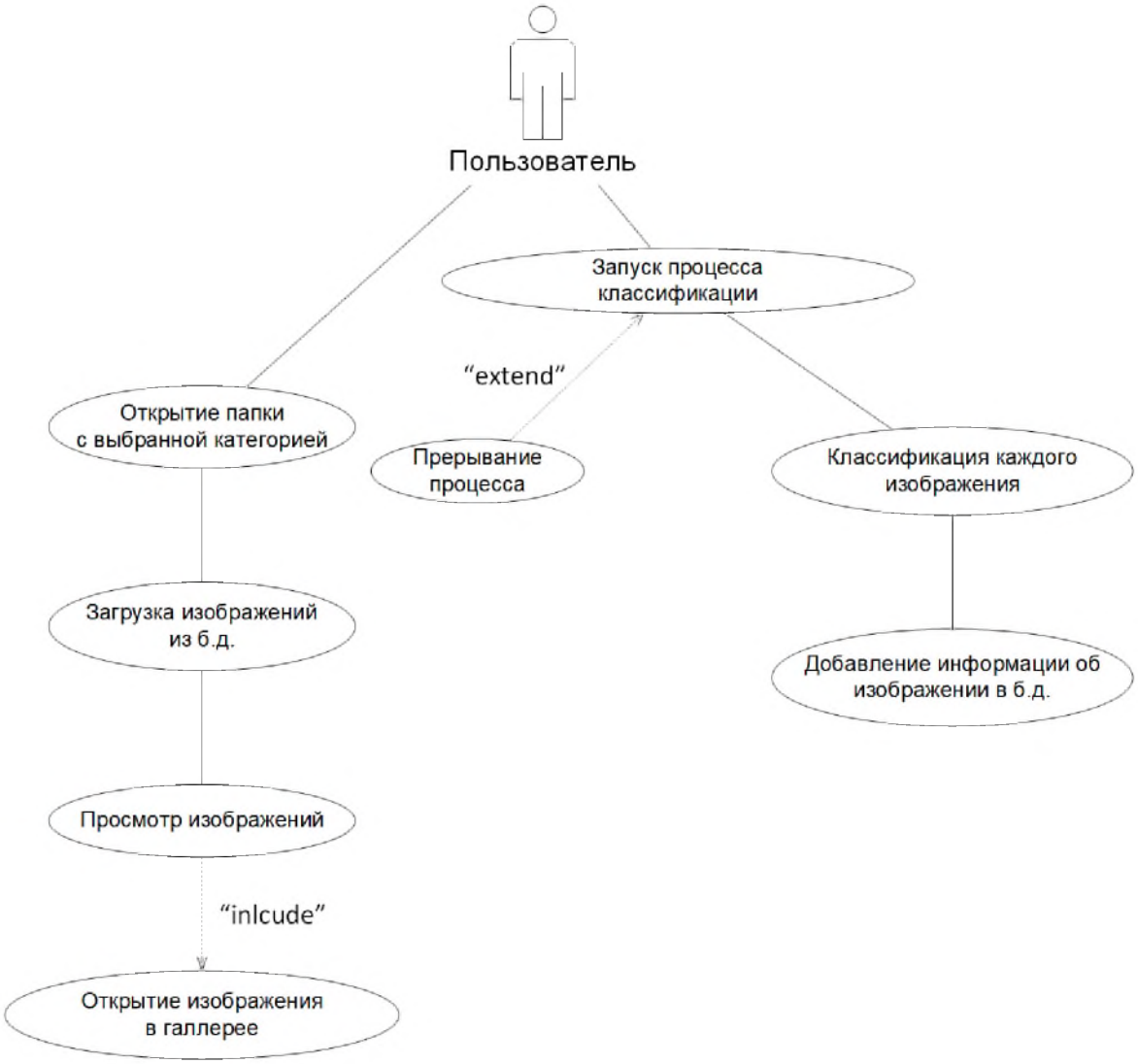


Рисунок 5 – UML-диаграмма вариантов использования

1.4 Архитектура приложения

Проектирование архитектуры кода является критически важным этапом разработки любого приложения для ОС Android. Некачественная архитектура создает серьезные проблемы: затрудняет процесс внесения изменений,

усложняет тестирование, ухудшает читаемость кода и общее понимание логики программы. Все это неизбежно приводит к росту количества ошибок и некорректной работе приложения в целом. Для того, чтобы избежать данных проблем, нужно следовать общим архитектурным принципам:

– разделение ответственности – код управления пользовательским интерфейсом и код моделей (бизнес-логики) должны быть разделены;

– управление состоянием UI из моделей. Модели – это компоненты, которые отвечают за обработку данных для приложения. Они не должны зависеть от элементов UI и компонентов ОС для исключения влияния на них жизненного цикла приложения.

Создание собственной архитектуры, которая будет следовать таким принципам – задача нелёгкая, которая требует большого количества опыта, но есть и готовые решения. Эти решения называются шаблонами проектирования. Одним из них является паттерн MVVM.

Паттерн MVVM позволяет разделить приложение на три функциональные части:

– Model – модель. В неё входят такие классы, как ImageClassifier.java и SQLitePhotoDatabase.java. Они отвечают за основную логику программы (классификация, работа с данными, хранение);

– View – представление, вид, пользовательский интерфейс приложения. К View относятся классы PermissionActivity.java, MainActivity.java, ImageActivity.java. Отвечает за отображение данных и взаимодействие с пользователем;

– ViewModel – модель представления. Включает в себя класс ViewModel.java, который служит прослойкой между View и Model. ViewModel предоставляет данные, которые View отображает, и обрабатывает действия пользователя.

На рисунке 6 показана взаимосвязь частей паттерна MVVM.

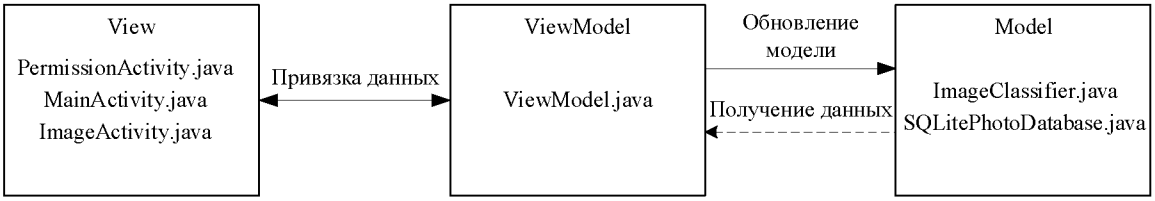


Рисунок 6 – Взаимодействие частей паттерна MVVM

1.5 Выбор языка программирования

Для создания приложений для Android-устройств используются два основных языка программирования: Java и Kotlin.

Java – это объектно-ориентированный язык программирования, созданный в 1995 году. Благодаря выбору компании Google, сыграл ключевую роль в разработке с самого начала мобильной ОС. За долгие годы разработчики Java создали обширную экосистему, полную библиотек, фреймворков и инструментов, которые способствуют более быстрым циклам разработки и ресурсоемкому решению проблем. Хотя за время существования Java появилось много новых языков, например Kotlin, ее стабильность, широкое распространение позволяют ей глубоко укорениться в ландшафте разработки Android, влияя на многие архитектурные и проектные шаблоны, используемые сегодня.

Kotlin – язык, представленный в 2011 году компанией JetBrains. Его разработали специально для замены Java, который считали слишком громоздким. Данный язык получился на 40% компактнее предшественника, что позволяет ускорить разработку приложений. При этом Kotlin полностью совместим с Java, легко интегрируется с существующими кодовыми базами Java, что позволяет разработчикам расширять имеющиеся библиотеки и знания. Эта совместимость сыграла важную роль в его принятии компанией Google, которая в 2017 году официально признала данный язык основным для разработки Android-приложений.

Для создания данного проекта был выбран язык программирования Java.

1.6 Выбор СУБД

В Android для хранения данных локально используется только одна реляционная БД – SQLite. Соответственно, она и будет ответственна за хранение информации об изображениях.

1.7 Выбор CNN-модели

При выборе подходящей CNN-модели для внедрения в мобильное приложение учитывались два ключевых критерия: размер модели и её точность. После анализа доступных решений были отобраны наиболее подходящие варианты, которые соответствуют заданным требованиям.

В таблице 2 приведено сравнение моделей. Точности Top-1 и Top-5 были получены на наборе данных ImageNet, который содержит 14 млн изображений 1000 классов.

Таблица 2 – Сравнение моделей

	Критерии		
Название модели	Размер	Точность Top-1	Точность Top-5
MobileNetV2	14 МБ	~72%	~89%
ResNet18-v2	44 МБ	~69%	~89%
SqueezeNet1.1	5 МБ	~56%	~79%
EfficientNet-Lite 4	51 МБ	~80%	~97%
ShuffleNet-v2	9 МБ	~69%	~88%

Исходя из вышеперечисленного сравнения, была выбрана модель MobileNetV2.

1.8 Выбор среды разработки

Для разработки приложений для Android-устройств существуют большое количество сред разработки, которые включают в себя все необходимые инструменты для создания программного обеспечения, такие как: текстовый редактор кода, отладчик, систему управления версиями и т.д. На данный момент существует 3 наиболее популярные среды разработки под Android:

– Eclipse;

– IntelliJ IDEA;

– Android Studio.

Eclipse – это бесплатная IDE, разработанная компанией Eclipse Foundation. Программа является основой, к которой подключаются различные модули, например, Java Development Tools и C/C++ Development Tools. На данный момент она не используется для разработки Android-приложений, т.к. Google прекратил поддержку в 2015 году. Используется только для поддержки старых проектов.

IntelliJ IDEA – это IDE от компании JetBrains, поддерживающая множество языков программирования, таких как Java, Kotlin, Python, JavaScript и т.п. Идеально подходит для разработки продуктов для ОС Android, т.к. у данной среды есть все необходимые инструменты: поддержка Android через плагин, система сборки Gradle. Имеет единственный недостаток – за IDE IntelliJ Idea нужно платить.

Android Studio – это официальная IDE для разработки Android-приложений, созданная Google на основе IntelliJ IDEA. Отличительные особенности этой IDE от предшественника: полная интеграция с Android SDK (поддержка всех версий Android), визуальный редактор макетов, наличие

встроенного эмулятора и глубокой поддержки Gradle (система сборки и управление зависимостями).

Из всех вышеперечисленных сред разработки, выбор пал именно на Android Studio ввиду того, что это IDE создано специально для создания приложений для Android.

Для обучения модели свёрточной нейронной сети используются два основных IDE – это Spyder и PyCharm.

Spyder – это бесплатная IDE для научных исследований, машинного обучения и анализа данных. Она создана с учётом специфики работы с Python и имеет все необходимые библиотеки для обучения нейронных сетей.

PyCharm – это интегрированная среда разработки для языка Python. Она разработана компанией JetBrains и используется более чем 50% разработчиками Python. Среда предлагает множество функций для написания, запуска, отладки кода, автоматического тестирования и машинного обучения. Есть две версии данного продукта: бесплатная – PyCharm Community Edition, и платная – PyCharm Professional Edition. Платная версия программы поддерживает библиотеки, необходимые для обучения нейронных сетей, что является главным минусом этой среды разработки.

Так как IDE Spyder является полностью бесплатным продуктом, имеющим все необходимые инструменты для разработки, в ней и будет проводиться обучение.

2 Разработка приложения

2.1 Обучение модели

Существует два способа решения проблемы классификации изображений: использование предварительно обученной модели, что не является рациональным подходом, так как модель обучалась на большом наборе данных для масштабной задачи классификации, или создание собственной модели для конкретной задачи с применением трансферного обучения.

Идея трансферного обучения заключается в том, чтобы на основе знаний, полученных моделью при решении одной задачи, построить свою модель для решения другой, но связанной задачи.

Основные принципы данного подхода:

– использование предобученной модели: берутся слои нейронной сети, обученной на большом наборе данных. Эти слои уже умеют извлекать общие признаки изображений (края, текстуры, формы);

– адаптация под новую задачу: заменяются или добавляются новые слои поверх предобученной части модели. Например, замена последних полносвязных слоёв на слои, соответствующие числу классов в новой задаче;

– заморозка базовых слоёв: веса предобученных слоёв замораживаются, чтобы они не изменялись во время обучения на новых данных. Это сохраняет уже выученные общие признаки и предотвращает переобучение;

– дообучение: обучаются только новые добавленные слои.

Рабочий процесс машинного обучения состоит из следующих этапов:

– создание тренировочного, валидационного и тестового наборов данных;

– создание базовой модели из предварительно обученной;

– обучение модели и тонкая настройка;

– оценка и экспорт.

2.1.1 Создание набора данных

В свободном доступе были найдены изображения 5 категорий: «Авто», «Документы», «Еда», «Животные», «Люди». Данный набор данных был загружен, и на его основе сформированы тренировочные и валидационные наборы. Код процедуры создания набора данных представлен листингом 1.

Листинг 1 – Создание набора данных

```
BATCH_SIZE = 32
IMG_SIZE = (224, 224)
directory = "C:\\Users\\iljak\\Desktop\\dataset"
train_dataset = image_dataset_from_directory(directory,
                                             shuffle=True,
                                             batch_size=BATCH_SIZE,
                                             image_size=IMG_SIZE,
                                             validation_split=0.2,
                                             subset='training',
                                             seed=42)
validation_dataset = image_dataset_from_directory(directory,
                                             shuffle=True,
                                             batch_size=BATCH_SIZE,
                                             image_size=IMG_SIZE,
                                             validation_split=0.2,
                                             subset='validation',
                                             seed=42)
```

Поскольку исходный набор данных не содержит тестового набора, он был создан из валидационного набора [5]. Код для создания данного набора представлен в листинге 2.

Листинг 2 – Создание тестового набора

```
val_batches = tf.data.experimental.cardinality(validation_datase
t)
test_dataset = validation_dataset.take(val_batches // 5)
validation_dataset = validation_dataset.skip(val_batches // 5)
```

2.1.2 Создание модели

Базовая модель создаётся на основе модели MobileNetV2, разработанной в Google. Она предварительно обучена на наборе данных ImageNet, большом наборе данных, состоящим из 1,4 млн изображений и 1000 классов. ImageNet – это набор данных для обучения исследованию с широким спектром категорий, таких как jackfruit и syringe. Эта база знаний поможет классифицировать изображения из нашего конкретного набора данных [5].

Прежде чем загрузить модель MobileNetV2 для использования в качестве базовой, необходимо выполнить нормализацию входных данных. Это критически важный шаг, поскольку модель ожидает входные значения пикселей в [-1, 1], но на данный момент значения пикселей в изображениях находятся в [0, 255]. Чтобы масштабировать их, используется метод предварительной обработки, включенный в модель [5]. В листинге 3 представлен код метода предварительной обработки.

Листинг 3 – Метод предварительной обработки

```
preprocess_input = tf.keras.applications.mobilenet_v2.preprocess
_input
```

Далее нужно выбрать, какой уровень MobileNetV2 будет использоваться для извлечения признаков. Самый последний слой классификации («сверху», поскольку большинство диаграмм моделей машинного обучения идут снизу вверх) не очень полезен. Вместо этого следует пользоваться общепринятой практикой, чтобы полагаться на самый последний слой перед операцией выравнивания. Этот слой называется «слоем узкого места». Функции слоя узкого места сохраняют большую общность по сравнению с конечным/верхним слоем [5].

Наконец, перейдём к созданию экземпляра модели MobileNetV2 с предварительно загруженными весами, обученными в ImageNet. Указав аргумент include_top=False, загружается сеть, которая не включает слои классификации вверху, что идеально подходит для извлечения признаков [5]. Код для загрузки предварительно обученной модели представлен листингом 4.

Листинг 4 – Загрузка предварительно обученной модели

```
IMG_SHAPE = IMG_SIZE + (3,)
base_model =
tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
                                  include_top=False,
                                  weights='imagenet')
```

Следующим шагом важно заморозить слои перед компиляцией и обучением модели. Замораживание (путем установки layer.trainable = False)

предотвращает обновление весов в данном слое во время обучения. MobileNetV2 имеет много слоев, поэтому установка флага trainable всей модели в значение False, заморозит их все [5]. В листинге 5 показан код для заморозки слоёв модели.

Листинг 5 – Заморозка слоёв модели

```
base_model.trainable = False
```

Кроме того, нужно добавить слои классификатора поверх слоёв базовой модели, которые и будут обучаться в дальнейшем. Одним из этих слоёв является слой глобального среднего пулинга, который используется для преобразования многомерных признаков, полученных на выходе предобученной модели, в компактный вектор. Кодом в листинге 6 и 7 показано добавление слоёв классификатора.

Листинг 6 – Добавление слоя классификатора

```
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
```

Следующим слоем будет использоваться полносвязный слой с 5 нейронами (количество нейронов определяется числом классов). Данный слой используется для преобразования усреднённых признаков в предсказания.

Листинг 7 – Добавление слоя классификатора

```
prediction_layer = tf.keras.layers.Dense(5, activation = 'softma
x')
prediction_batch = prediction_layer(feature_batch_average)
```

Объединив вместе слои увеличения данных, масштабирования, предобученной модели и классификатора, получим собственную модель [5]. Код создания модели представлен листингом 8.

Листинг 8 – Создание модели

```
inputs = tf.keras.Input(shape=(224, 224, 3))
x = preprocess_input(inputs)
x = base_model(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)
```

2.1.3 Обучение модели

Перед обучением новую модель нужно скомпилировать. Код в листинге 9 показывает компиляцию модели.

Листинг 9 – Компиляция модели

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=
                                    base_learning_rate),
                                    loss=tf.keras.losses.SparseCat
                                    egoricalCrossentropy(from_logi
                                    ts=False),
                                    metrics=['accuracy'])
```

Обучение происходит в течение 10 эпох. В листинге 10 представлен код обучения модели.

Листинг 10 – Добавление слоёв классификатора

```
initial_epochs = 10
history = model.fit(train_dataset,
                    epochs=initial_epochs,
                    validation_data=validation_dataset)
```

Кривые обучения точности/потери при обучении и проверке показаны на рисунке 7.

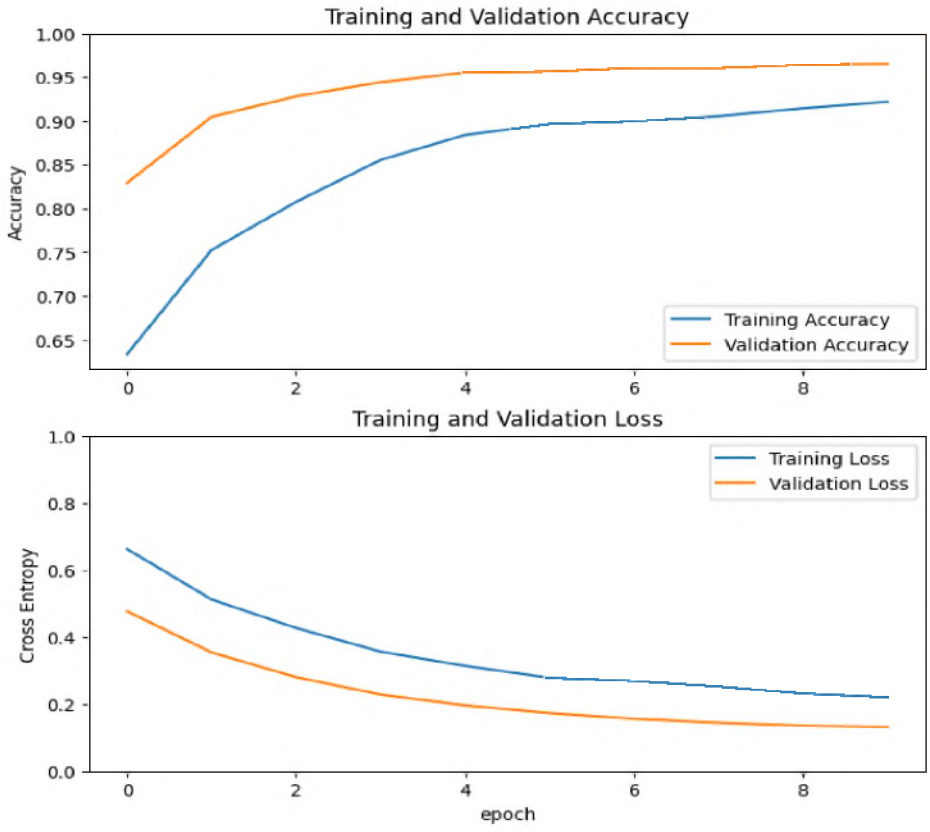


Рисунок 7 – Кривые обучения точности/потери

2.1.4 Тонкая настройка

Одним из способов ещё большего повышения производительности является обучение (или «точная настройка») весов верхних слоев предварительно обученной модели наряду с обучением добавленного классификатора [5].

В большинстве свёрточных сетей чем выше уровень, тем более он специализирован. Первые несколько слоев изучают очень простые и общие функции, которые применимы практически ко всем типам изображений. Цель тонкой настройки — адаптировать специализированные функции для работы с новым набором данных, а не перезаписывать общее обучение [5].

Для начала нужно разморозить слои базовой модели и сделать нижние слои необучаемыми. В листинге 11 показан код данной процедуры.

Листинг 11 – Разморозка слоёв

```
base_model.trainable = True
fine_tune_at = 100
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable = False
```

Осталось лишь скомпилировать модель и повторно обучить, используя при этом более низкую скорость обучения. В противном случае модель может очень быстро переобучиться [5]. Код процедуры компиляции и обучения модели представлен листингом 12.

Листинг 12 – Компиляция и повторное обучение модели

```
model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy
                                        (from_logits=
                                        False),
                                        optimizer =
                                        tf.keras.optimizers.RMSpr
                                        op(learning_rate=base_lea
                                        rning_rate/10),
                                        metrics=['accuracy'])
fine_tune_epochs = 10
total_epochs =  initial_epochs + fine_tune_epochs
history_fine = model.fit(train_dataset,
                         epochs=total_epochs,
                         initial_epoch=history.epoch[-1],
                         validation_data=validation_dataset)
```

2.1.5 Оценка и экспорт модели

После вышеперечисленных манипуляций можно приступить к использованию модели на тестовом наборе. В листинге 12 продемонстрирован код запуска модели.

Листинг 12 – Запуск модели

```
loss, accuracy = model.evaluate(test_dataset)
```

Результаты предсказаний показаны на рисунках 8 – 13.



Рисунок 8 – Результат предсказания №1

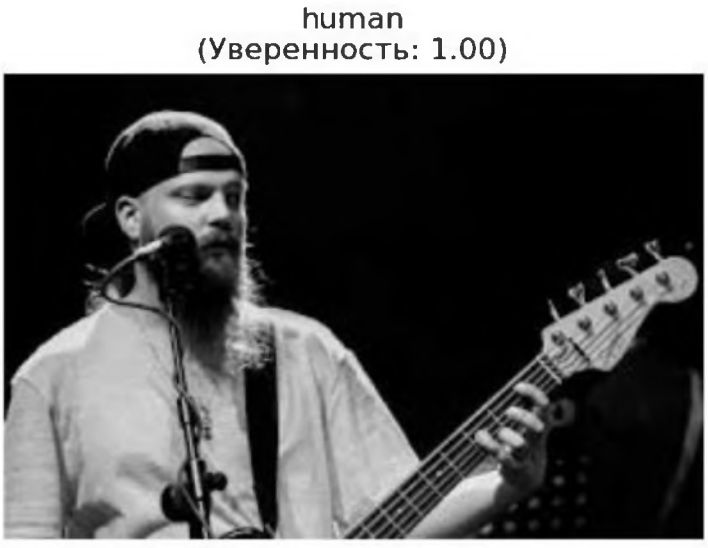


Рисунок 9 – Результат предсказания №2

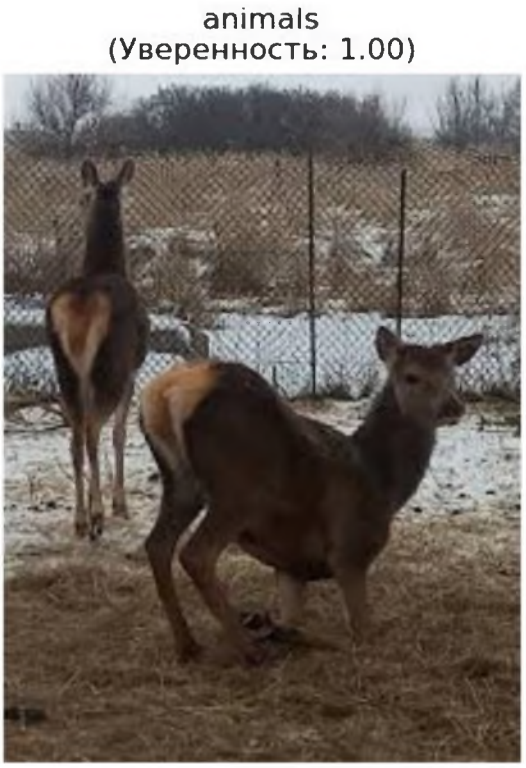


Рисунок 10 – Результат предсказания №3



Рисунок 11 – Результат предсказания №4



Рисунок 12 – Результат предсказания №5



Рисунок 13 – Результат предсказания №6

Экспорт модели происходит путём конвертации её в формат TFLite. TensorFlow Lite (TFLite) – это формат и набор инструментов, разработанные Google для запуска моделей машинного обучения на устройствах с ограниченными ресурсами. Предназначен для смартфонов (Android, iOS), микроконтроллеров (TensorFlow Lite Micro) и других встроенных систем. Его

ключевые особенности – уменьшение размера модели и увеличение скорости вывода.

Сам процесс конвертации состоит из двух этапов: экспорт модели в формат «saved_model» и последующая конвертация из этого формата в TFLite. Код процесса конвертации представлен в листинге 13.

Листинг 13 – Экспорт и конвертация в формат TFLite

```
model.export("C:\\Users\\iljak\\Downloads\\model")
converter = tf.lite.TFLiteConverter.from_saved_model("C:\\Users\
\iljak\\Downloads\\model\\")
tflite_model = converter.convert()
with open('C:\\Users\\iljak\\Downloads\\model\\mobilenet.tflite',
'wb') as f:
    f.write(tflite_model)
```

2.2 Разработка компонентов приложения

Паттерн MVVM, как уже описывалось ранее, обеспечивает чёткое разделение ответственности между компонентами. Данный паттерн делит проект на 3 части: Model, View, ViewModel. Каждый из трёх слоёв выполняет строго определенные задачи, что минимизирует связанность между ними и повышает модульность приложения.

2.2.1 Разработка Model

Слой Model представляет собой ядро приложения, отвечающее за работу с данными и бизнес-логику. Он включает в себя три ключевых класса: SQLitePhotoDatabase.java, ImageClassifier.java, UserImage.java.

Класс SQLitePhotoDatabase нужен для создания и последующего обновления локальной базы данных для хранения информации об изображениях. База данных состоит лишь из одной таблицы, которая имеет следующие поля:

– _id (INTEGER, PRIMARY KEY) – уникальный идентификатор изображения, совпадающего с ID изображения в системной галерее (MediaStore). Это позволяет избежать дублирования данных и упрощает синхронизацию;

– name (TEXT) – название файла изображения;

– uri (TEXT) – URI изображения в формате строки;

– result (TEXT) – категория, присвоенная изображению после классификации;

– confidence (TEXT) – уровень уверенности модели в процентах.

На рисунке 14 показана ER-диаграмма.

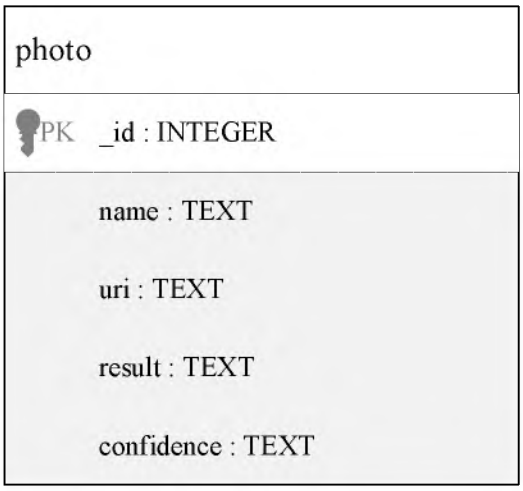


Рисунок 14 – ER-диаграмма

Данный класс имеет несколько методов: «onCreate», «findNewImages», «findProcessedImages», «getImgWithCategory».

Метод «onCreate» инициализирует базу данных при первом запуске приложения, создавая таблицу «photo» с указанными выше полями. Код метода «onCreate» представлен листингом 14.

Листинг 14 – Метод «onCreate»

```
@Override
    public void onCreate(SQLiteDatabase db) {
        String query = "CREATE TABLE " + TABLE_NAME + " ("
                + ID + " INTEGER PRIMARY KEY, "
                + NAME + " TEXT, " + URI + " TEXT, " + RESULT +
" TEXT, " + CONFIDENCE + " TEXT);";
        db.execSQL(query);
    }
```

Метод «findNewImages» выполняет поиск новых изображений, которые еще не были обработаны приложением. Он сравнивает изображения из системной галереи (MediaStore) с данными в локальной базе данных (SQLite), чтобы определить, какие изображения нужно классифицировать. Код метода «findNewImages» показан в листинге 15.

Листинг 15 – Метод «findNewImages»

```
public Cursor findNewImages(Cursor cursorWithProcImg) {
        Set<Long> existingIds = new HashSet<>();
        if (cursorWithProcImg != null &&
cursorWithProcImg.moveToFirst()) {
            do {
                long id =
cursorWithProcImg.getLong(cursorWithProcImg.getColumnIndexOrThro
w("_id"));
                existingIds.add(id);
            } while (cursorWithProcImg.moveToNext());
        }
        String[] projection = {
                MediaStore.Images.Media._ID,
                MediaStore.Images.Media.DISPLAY_NAME,
        };

        Cursor mediaStoreCursor =
context.getContentResolver().query(
                MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
                projection,
                null,
                null,
                MediaStore.Images.Media.DATE_TAKEN + " DESC"
        );
        MatrixCursor resultCursor = new MatrixCursor(new
String[]{
                MediaStore.Images.Media._ID,
                MediaStore.Images.Media.DISPLAY_NAME,
                "uri"
        });
        if (mediaStoreCursor != null) {
            try {
                while (mediaStoreCursor.moveToNext()) {
                    long id =
mediaStoreCursor.getLong(mediaStoreCursor.getColumnIndexOrThrow(
MediaStore.Images.Media._ID));
                    if (!existingIds.contains(id)) {
                        String name =
mediaStoreCursor.getString(mediaStoreCursor.getColumnIndexOrThro
w(MediaStore.Images.Media.DISPLAY_NAME));
```

```
                        Uri imageUri =
ContentUris.withAppendedId(MediaStore.Images.Media.EXTERNAL_CONT
ENT_URI, id);
                        SQLiteDatabase dbWriter =
this.getWritableDatabase();
                        ContentValues values = new
ContentValues();
                        values.put("_id", id);
                        values.put("name", name);
                        values.put("uri", imageUri.toString());
                        dbWriter.insertWithOnConflict("photo",
null, values, SQLiteDatabase.CONFLICT_REPLACE);
                        resultCursor.addRow(new Object[]{id,
name, imageUri.toString()});
                    }
                }
            } finally {
                mediaStoreCursor.close();
            }
        }
        return resultCursor;
    }
```

Метод «findProcessedImages» возвращает список с изображениями из локальной базы данных, где есть результат классификации. Код метода представлен листингом 16.

Листинг 16 – Метод «findProcessedImages»

```
public Cursor findProcessedImages(){
        SQLiteDatabase dbReader = this.getReadableDatabase();
        String selection = "result IS NOT NULL";
        return dbReader.query(
                "photo", null, selection,null,null,null,null
        );
    }
```

Метод «getImgWithCategory» возвращает список с изображениями одной категории. В листинге 17 продемонстрирован код данного метода.

Листинг 17 – Метод «getImgWithCategory»

```
public Cursor getImgWithCategory(String category){
        SQLiteDatabase dbReader = this.getReadableDatabase();
        String selection = null;
        switch (category){
            case "Авто": selection = "result IS 'vehicle'";
break;
            case "Документы": selection = "result IS
'document'"; break;
            case "Еда": selection = "result IS 'food'"; break;
```

```
            case "Животные": selection = "result IS 'animals'";
break;
            case "Люди": selection = "result IS 'human'"; break;
            case "Не определено": selection = "result IS 'Не
определено'"; break;
        }
        return dbReader.query(
                "photo", null, selection,null,null,null,null
        );
    }
```

Класс ImageClassifier отвечает за классификацию каждого изображения с помощью модели TensorFlow Lite.

В этом классе есть следующие методы: «loadModel», «loadLabels», «classify».

Метод «loadModel» загружает модель TensorFlow Lite из файла mobilenetV2.tflite, расположенного в активах приложения. Код метода представлен листингом 18.

Листинг 18 – Метод «loadModel»

```
private MappedByteBuffer loadModel(Context context) throws
IOException {
        AssetFileDescriptor fileDescriptor =
context.getAssets().openFd(MODEL_PATH);
        FileInputStream inputStream = new
FileInputStream(fileDescriptor.getFileDescriptor());
        FileChannel fileChannel = inputStream.getChannel();
        long startOffset = fileDescriptor.getStartOffset();
        long declaredLength =
fileDescriptor.getDeclaredLength();
        return fileChannel.map(FileChannel.MapMode.READ_ONLY,
startOffset, declaredLength);
    }
```

Следующий метод «loadLabels» считывает метки категорий из файла «labels.txt», который содержит 5 классов. Листинг 19 показывает код данного метода.

Листинг 19 – Метод «loadLabels»

```
private List<String> loadLabels(Context context) throws
IOException {
        List<String> labelList = new ArrayList<>();
        BufferedReader reader = new BufferedReader(new
InputStreamReader(context.getAssets().open(LABEL_PATH)));
        String line;
        while ((line = reader.readLine()) != null) {
```

```
            labelList.add(line);
        }
        reader.close();
        return labelList;
    }
```

Основную работу выполняет метод «classify». Он загружает изображение по URI из галереи устройства. Сжимает его до размера 224x224 пикселей, создаёт тензор и запускает классификацию с помощью метода «classifyWithTFLite», который выполняет инференс модели и возвращает массив уверенностей для каждого класса. После из массива выбирается категория с наибольшей вероятностью, и данный результат (категория и уверенность) сохраняется в базу данных. Код методов «classify» и «classifyWithTFLite» продемонстрирован листингом 20 и 21, соответственно.

Листинг 20 – Метод «classify»

```
public void classify(Uri imageUri){
        Bitmap bitmap = null;
        try {
            bitmap =
MediaStore.Images.Media.getBitmap(context.getContentResolver(),
imageUri);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
        Bitmap image = bitmap.copy(Bitmap.Config.ARGB_8888,
true);
        TensorImage tensor = new TensorImage(DataType.FLOAT32);
        tensor.load(image);
        TensorImage processedTensor =
imageProcessor.process(tensor);
        float[] output = classifyWithTFLite(processedTensor);
        int maxIndex = 0;
        float maxConfidence = 0;
        for (int i = 0; i < output.length; i++) {
            if (output[i] > maxConfidence) {
                maxConfidence = output[i];
                maxIndex = i;
            }
        }
        SQLiteDatabase db =
databaseHelper.getWritableDatabase();
        ContentValues contentValues = new ContentValues();
        if (maxConfidence < 0.75) {
            contentValues.put("result", "Не определено");
        }
```

```
        else {
            contentValues.put("result", labels.get(maxIndex));
        }
        contentValues.put("confidence", "(" + (maxConfidence *
100) + "%)");
        db.update("photo", contentValues, "uri = \"" + imageUri
+ "\";", null);
    }
```

Листинг 21 – Метод «classifyWithTFLite»

```
private float[] classifyWithTFLite(TensorImage tensorImage) {
        int[] outputShape =
interpreter.getOutputTensor(0).shape();
        FloatBuffer outputBuffer =
FloatBuffer.allocate(outputShape[1]);
        outputBuffer.rewind();

interpreter.run(tensorImage.getTensorBuffer().getBuffer(),
outputBuffer);
        outputBuffer.rewind();
        float[] output = new float[outputBuffer.capacity()];
        outputBuffer.get(output);
        return output;
    }
```

2.2.2 Разработка View

Слой View (классы PermissionActivity, MainActivity, ImageActivity) отвечает за взаимодействие с пользователем, предоставляя интуитивно понятный интерфейс для управления приложением и отображения результатов. Он реализован через активности и адаптеры, которые тесно интегрированы с ViewModel для обеспечения реактивности.

PermissionActivity встречает пользователя при запуске приложения. Главная задача данной активности – получить разрешение от пользователя для доступа к файлам. Это разрешение необходимо для дальнейшей работы приложения.

Главным экраном приложения является класс MainActivity. Он отображает папки с категориями, статистику обработанных/необработанных изображений, а также кнопки управления.

Роль ImageActivity заключается в отображении изображений выбранной категории в виде сетки.

Помимо классов активностей есть 2 вспомогательных класса: класс FolderAdapter, который нужен для отображения папок с категориями, и класс ImageAdapter, который отображает изображения.

2.2.3 Разработка ViewModel

Класс ViewModel выступает центральным элементом архитектуры приложения, выполняя роль посредника между Model и View. Он не только управляет бизнес-логикой, но и обеспечивает реактивное обновление интерфейса.

ViewModel решает несколько ключевых задач:

– координирует загрузку и синхронизацию данных между локальной БД (SQLitePhotosDatabase) и системным хранилищем (MediaStore);

– обрабатывает действия пользователя (например, нажатие кнопки "Старт") и запускает выполнение операций;

– использует LiveData для автоматического обновления UI при изменении данных.

Основные методы: «findProcessedImages», «findNewImages», «classify», «stopExecutor», «onCleared».

Метод «findProcessedImages» загружает с помощью Model список изображений, уже прошедших классификацию, и отправляет в View их количество. Код данного метода показан в листинге 22.

Листинг 22 – Метод «findProcessedImages»

```
public void findProcessedImages() {
        if (cursorWithProcImg != null) {
            cursorWithProcImg.close();
        }
        cursorWithProcImg =
databaseHelper.findProcessedImages();
        procImgSize.postValue(cursorWithProcImg.getCount());
    }
```

Метод «findNewImages», обращаясь к Model, идентифицирует новые изображения и также отправляет в View их количество. Листинг 23 демонстрирует код данного метода.

Листинг 23 – Метод «findNewImages»

```
public void findNewImages() {
        if (cursorWithNewImg != null) {
            cursorWithNewImg.close();
        }
        cursorWithNewImg =
databaseHelper.findNewImages(cursorWithProcImg);
        newImgSize.postValue(cursorWithNewImg.getCount());
    }
```

Метод «classify» запускает процесс классификации новых изображений в фоновых потоках. Для каждого изображения создаётся отдельный поток (количество потоков равно половине количества ядер телефона). Также метод отправляет в View текущий прогресс обработки. Код показан в листинге 24.

Листинг 24 – Метод «classify»

```
public Boolean classify() {
       if (cursorWithNewImg == null ||
cursorWithNewImg.getCount() == 0) {
           return false;
       }
       isRunning.postValue(true);
       processedCount.set(0);
       if (executor.isShutdown()){
           executor =
Executors.newFixedThreadPool(Runtime.getRuntime().availableProce
ssors() / 2);
       }
       List<Long> imageIds = new ArrayList<>();
       cursorWithNewImg.moveToFirst();
       do {
           long id =
cursorWithNewImg.getLong(cursorWithNewImg.getColumnIndexOrThrow(
MediaStore.Images.Media._ID));
           imageIds.add(id);
       } while (cursorWithNewImg.moveToNext());
       for (Long id : imageIds) {
           executor.execute(() -> {
               try {
imageClassifier.classify(ContentUris.withAppendedId(MediaStore.I
mages.Media.EXTERNAL_CONTENT_URI, id));
               } catch (Exception ex){
                   Log.e("Thread", ex.getMessage());
               } finally {
                   int currentValue =
processedCount.incrementAndGet();
                   processValue.postValue(currentValue);
                   if (currentValue == imageIds.size()) {
                       new
Handler(Looper.getMainLooper()).post(() -> {
                           findProcessedImages();
```

```
                            findNewImages();
                            isRunning.postValue(false);
                        });
                    }
                }
            });
        }
        return true;
    }
```

Метод «stopExecutor» прерывает выполнение классификации и освобождает ресурсы. Листинг 25 демонстрирует код данного метода.

Листинг 25 – Метод «stopExecutor»

```
public void stopExecutor(){
        findProcessedImages();
        findNewImages();
        isRunning.postValue(false);
        executor.shutdownNow();
    }
```

Последний метод «onCleared» освобождает ресурсы при уничтожении ViewModel. Его код показан в листинге 26.

Листинг 26 – Метод «onCleared»

```
@Override
    protected void onCleared(){
        executor.shutdownNow();
        imageClassifier.close();
        databaseHelper.close();
        if (cursorWithProcImg != null) {
            cursorWithProcImg.close();
        }
        if (cursorWithNewImg != null) {
            cursorWithNewImg.close();
        }
    }
```

3 Тестирование приложения

Приложение встречает пользователя запросом на получение разрешения доступа к фото и мультимедиа устройства. На рисунке 15 показан данный запрос.

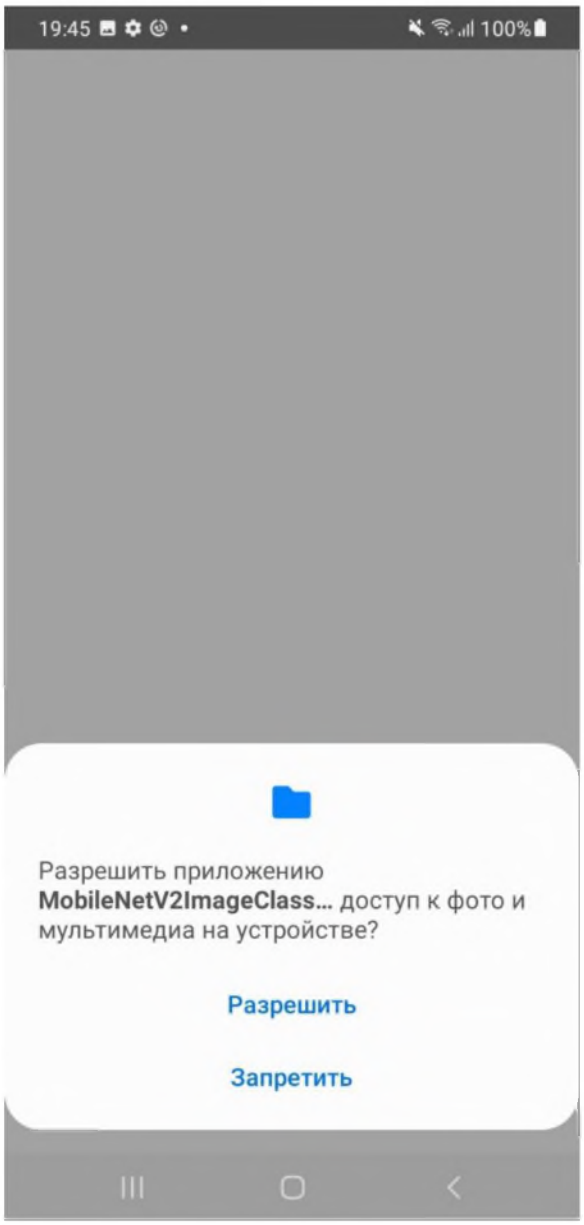


Рисунок 15 – Запрос разрешения

Ожидаемый результат при нажатии на клавишу «Разрешить» – переход к главной активности приложения. Результат данного действия представлен на рисунке 16.

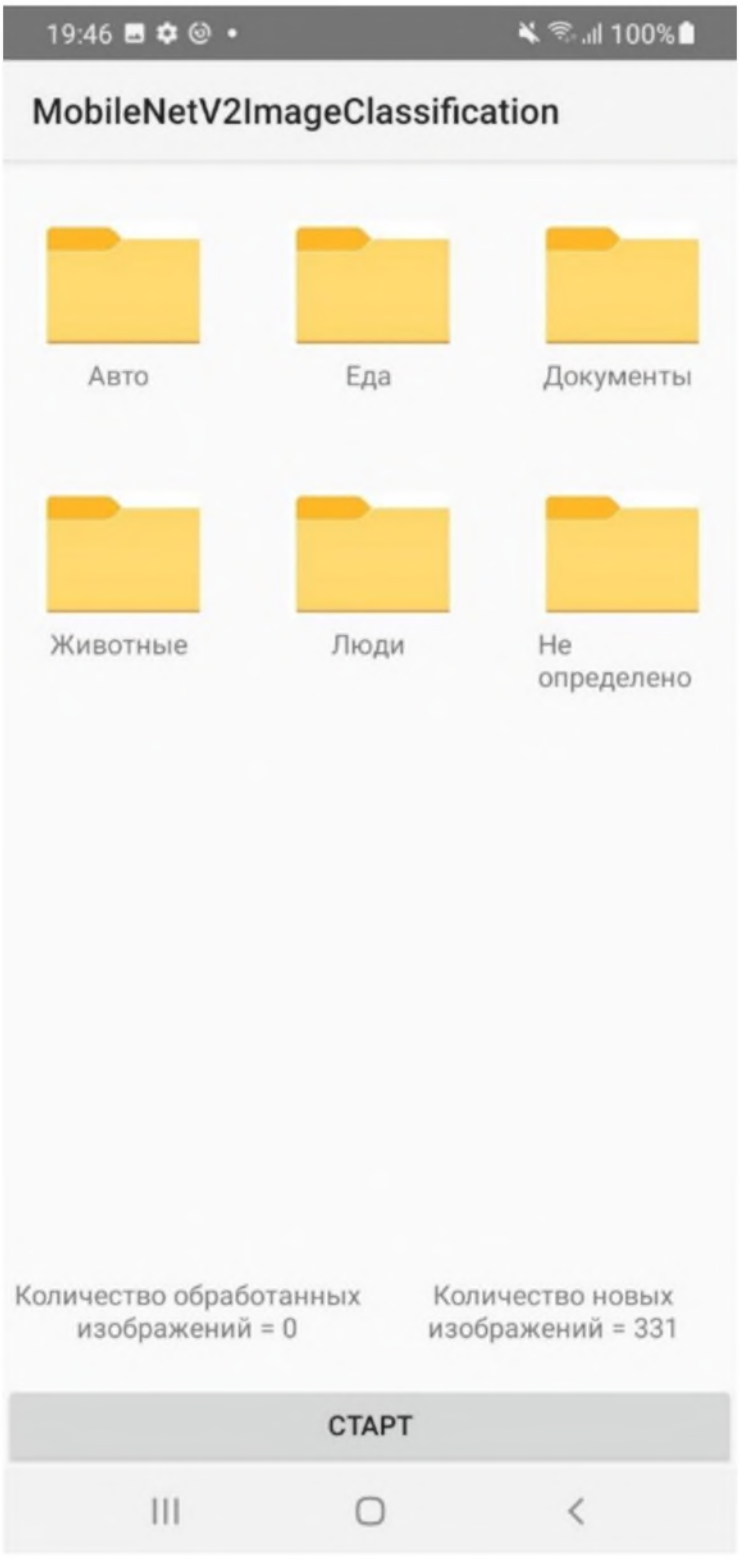


Рисунок 16 – Главный экран приложения

При нажатии на кнопку «Запретить» результат должен быть следующим: пользователю высветится уведомление о необходимости данного разрешения. На рисунке 17 показано уведомление.

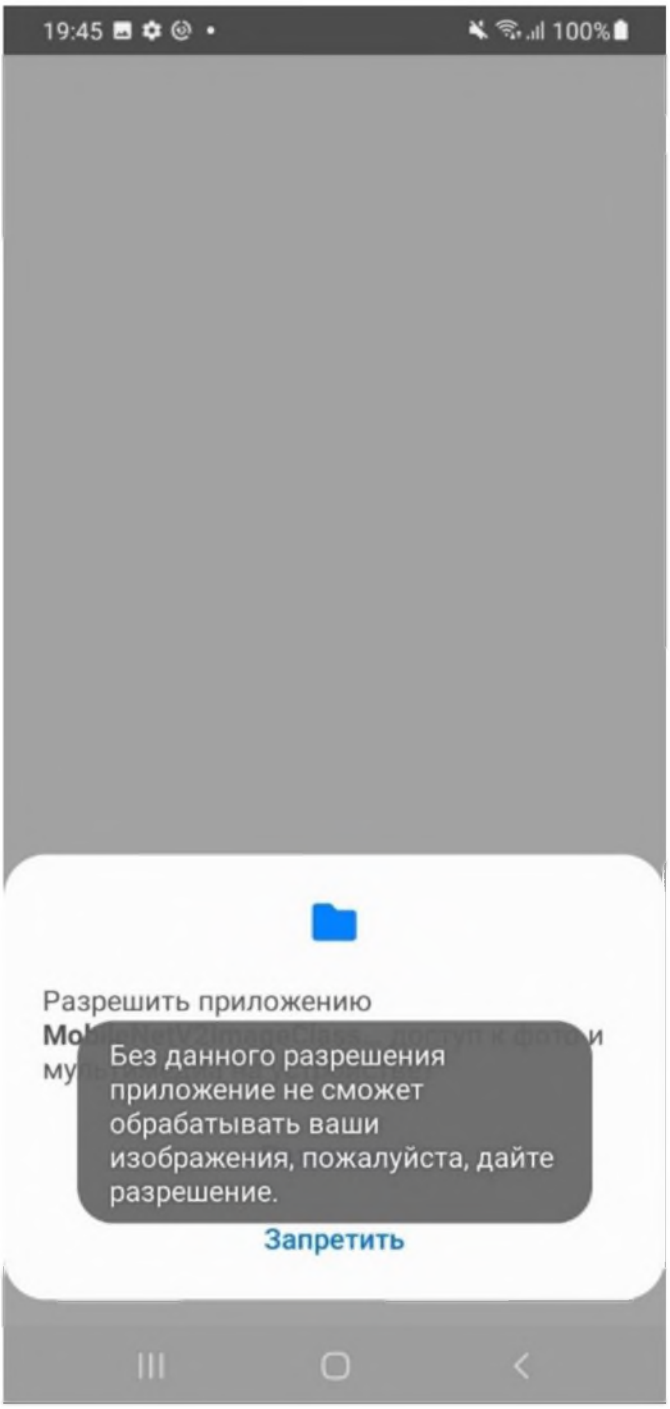


Рисунок 17 – Уведомление о необходимости разрешения

В главной активности должно быть видно количество новых изображений. Это свидетельствует тому, что у приложения есть все разрешения для получения пользовательских изображений, и оно может приступить к классификации. На рисунке 18 можно увидеть подтверждение выше сказанному.

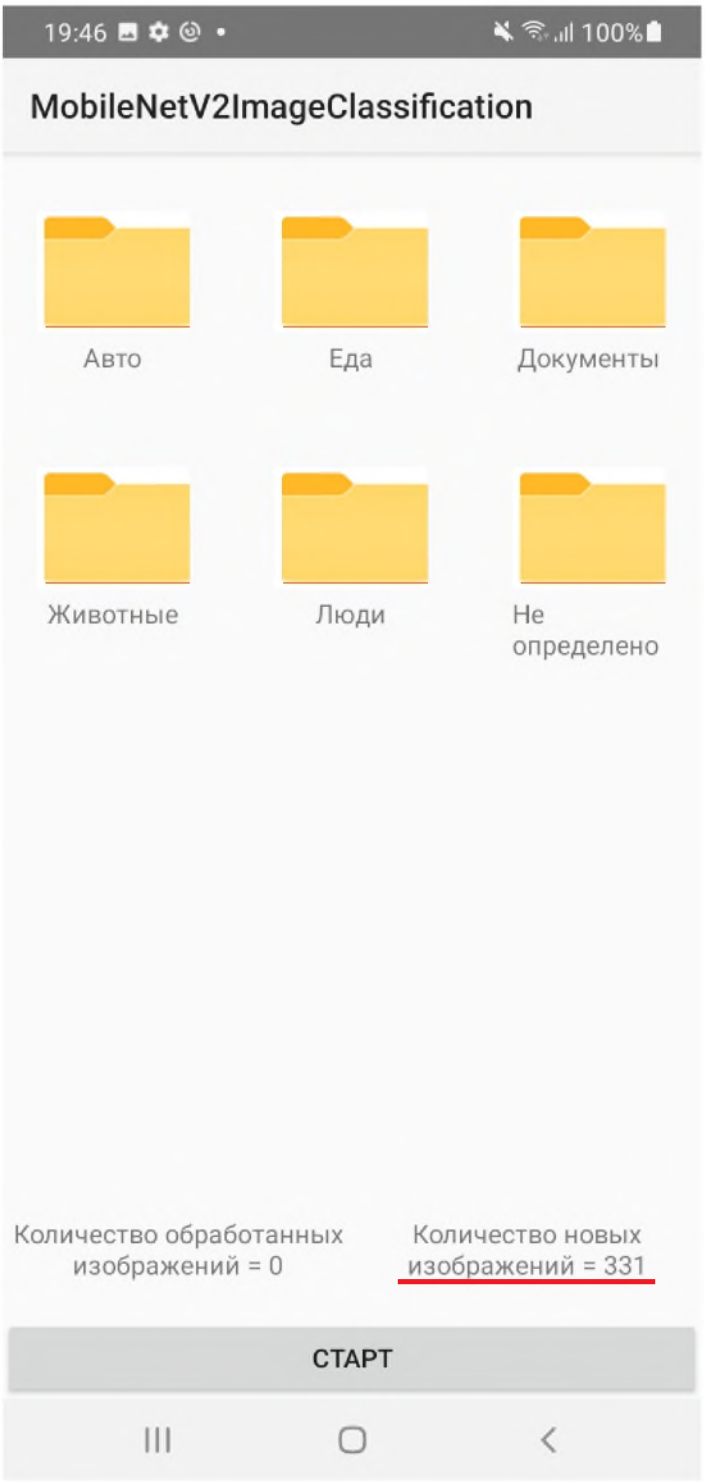


Рисунок 18 – Главное меню приложения

При нажатии на клавишу «Старт» перед пользователем появляется всплывающее окно, показывающее начало процесса классификации. Оно состоит из индикатора прогресса и клавиши «Стоп». На рисунке 19 демонстрируется вид данного окна.

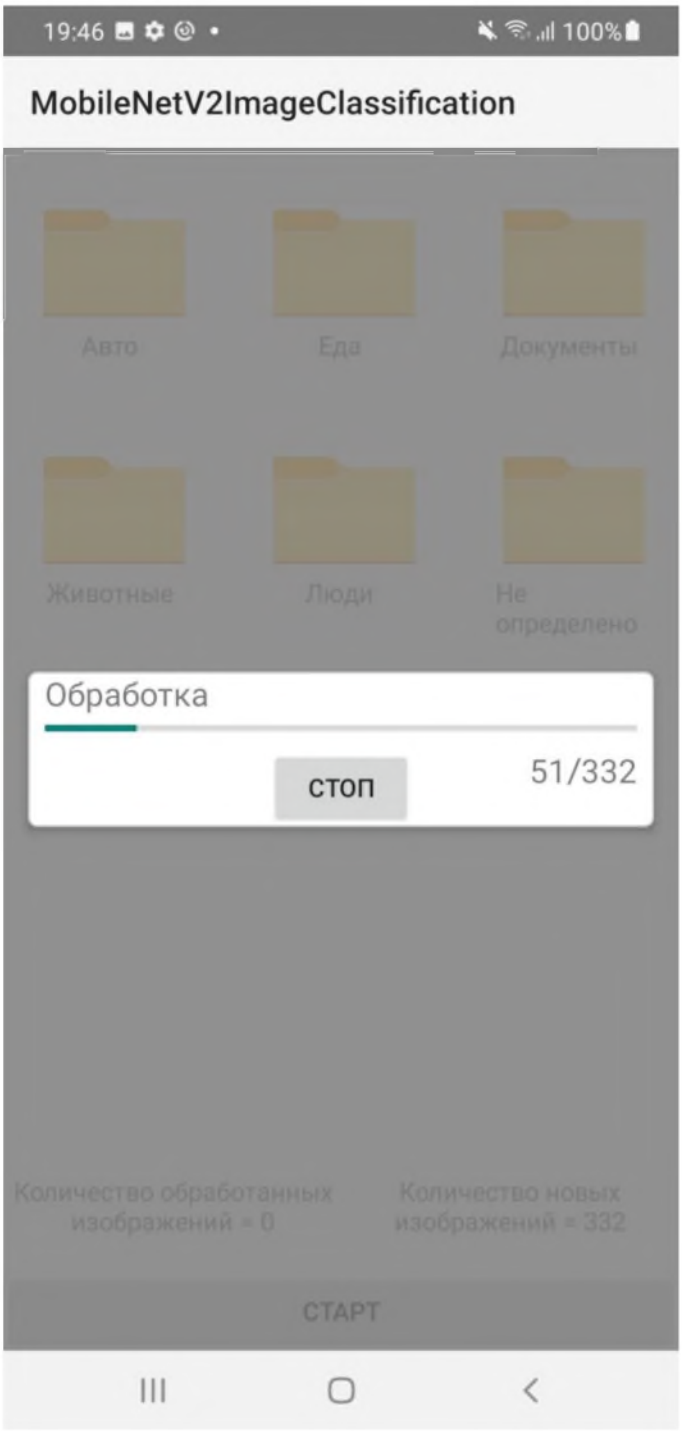


Рисунок 19 – Окно обработки

При нажатии на клавишу «Стоп» обработка прерывается, вплывающее окно закрывается, перед пользователем вновь появляется главная активность. Но прогресс обработки сохраняется: количество обработанных фотографий увеличивается, а необработанных уменьшается. Рисунок 20 показывает изменение количества изображений двух групп.

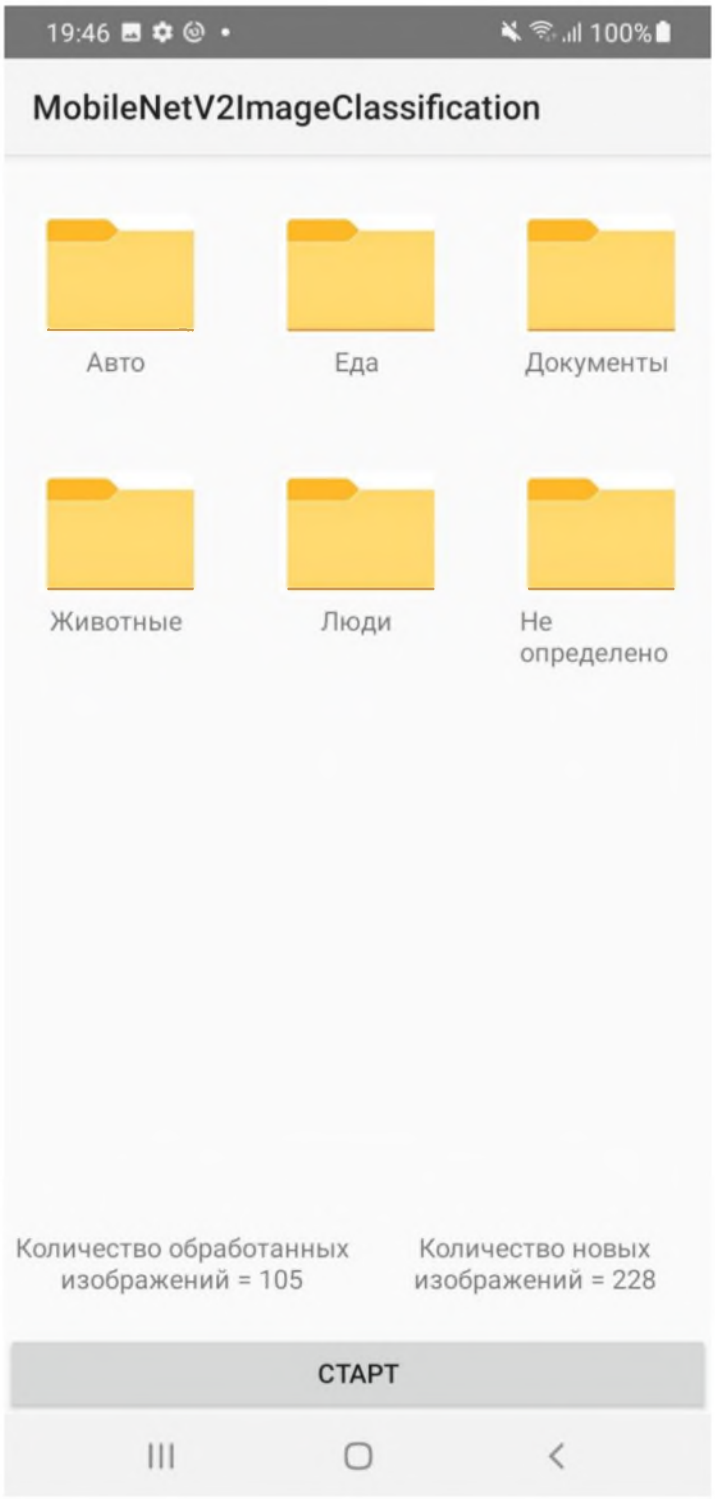


Рисунок 20 – Результат прерванной обработки

Если же не прерывать процесс обработки, то окно автоматически закрывается по окончании классификации. Количество обработанных фотографий достигает максимума. На рисунке 21 показано завершение процесса обработки.

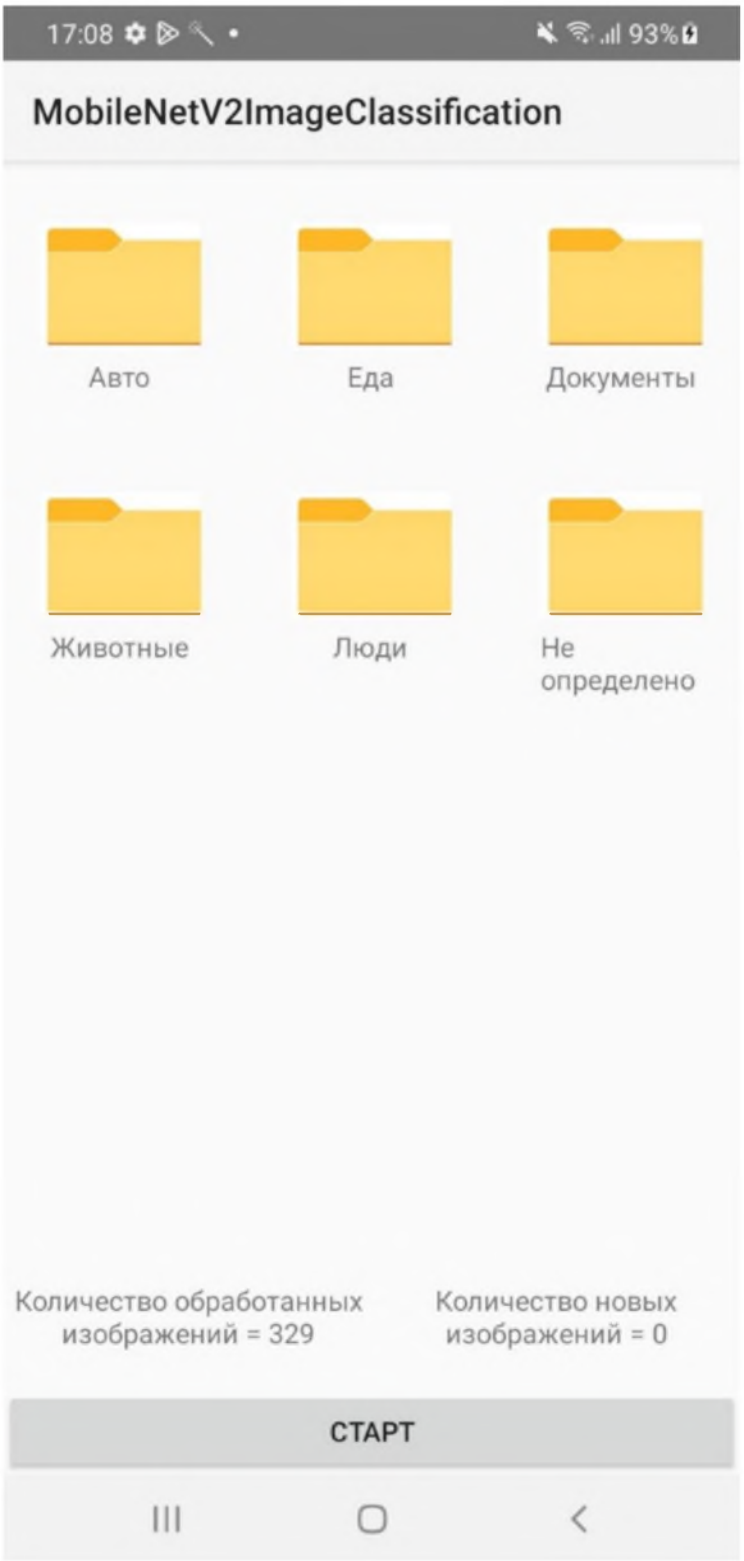


Рисунок 21 – Завершение процесса классификации

Если же попытаться начать процесс обработки без поступления новых изображений, то перед пользователем высветится уведомление об ошибке. Данное уведомление показано на рисунке 22.

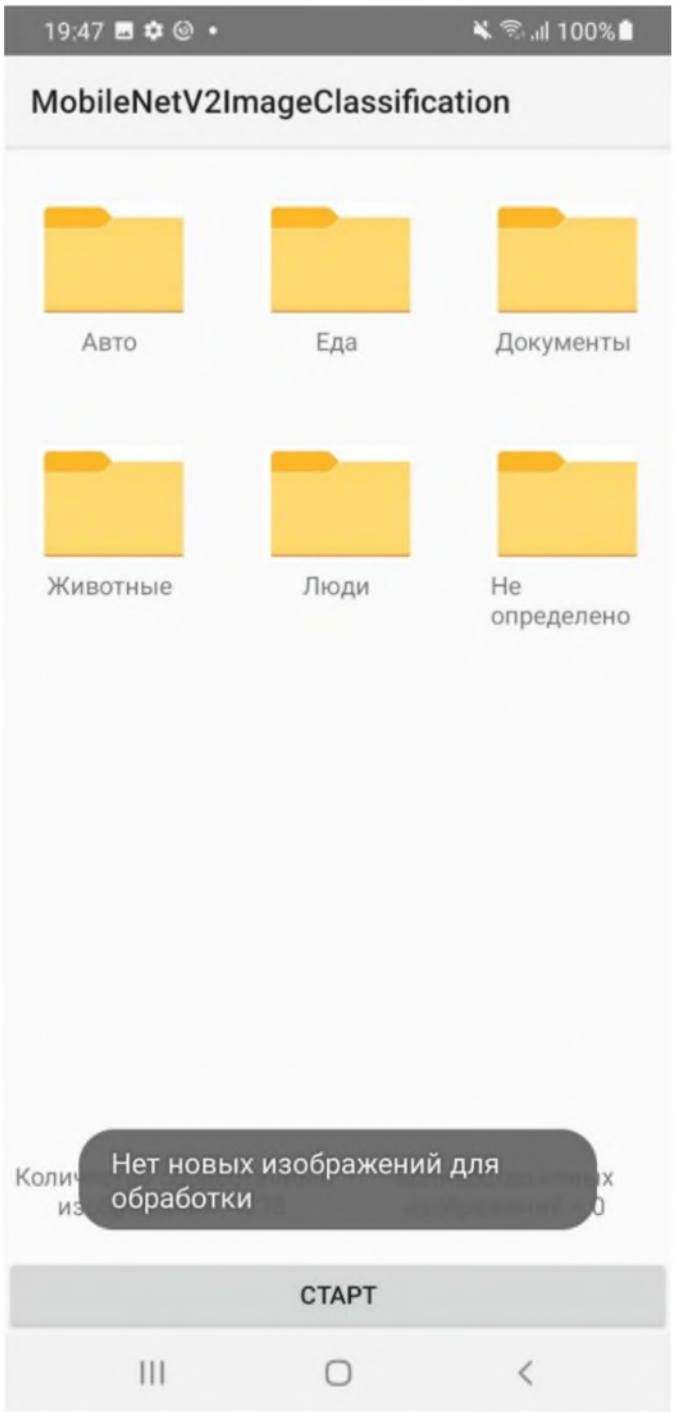


Рисунок 22 – Уведомление об ошибке

Для перехода к просмотру содержимого любой папки нужно нажать на иконку выбранной категории. После нажатия происходит переход к новой активности, в которой можно увидеть все изображения, относящиеся к данной категории, в виде сетки. На рисунке 23 можно увидеть изображения категории «Авто».

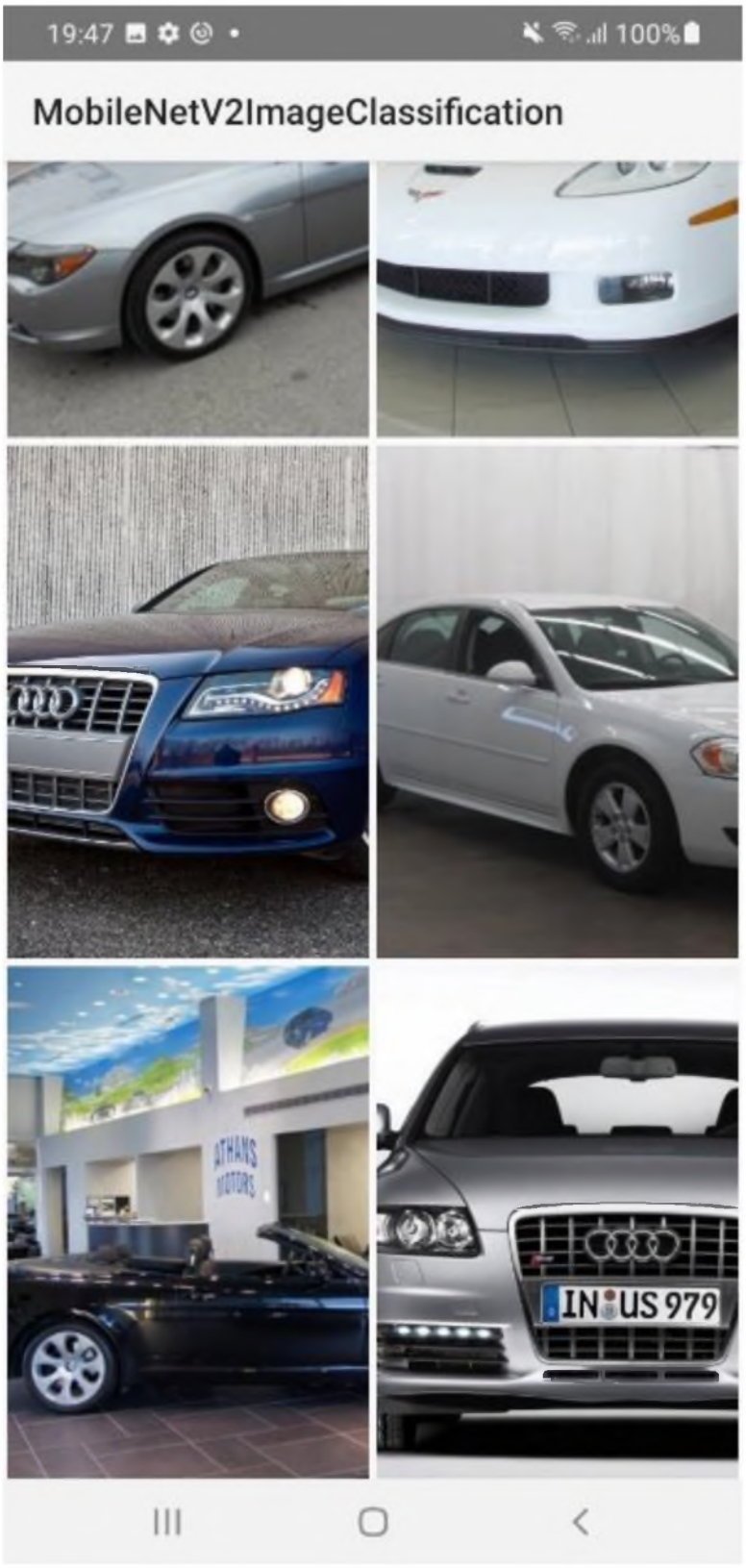


Рисунок 23 – Изображения категории «Авто»

При нажатии на любое изображение оно открывается в системной галерее. На рисунке 24 показано изображение в галерее.



Рисунок 24 – Открытое в галерее изображение

4 Безопасность жизнедеятельности

4.1 Правила безопасности при использовании смартфона

Чтобы не повредить телефон, аксессуары и подключенные устройства, а также снизить риск возгорания или поражения электрическим током, соблюдайте приведенные ниже меры предосторожности:

– не разбирайте телефон, зарядное устройство или аксессуары. Попытки вскрытия могут привести к поломке;

– избегайте механических воздействий: не сгибайте, не бросайте, не протыкайте корпус и не прикладывайте избыточное давление;

– не замыкайте контакты и держите устройство вдали от открытого огня, воды, жидкостей и источников тепла (например, духовок, микроволновых печей).

Правила зарядки:

– используйте сертифицированные источники питания;

– не заряжайте устройство на мягких поверхностях, вблизи воды или при повышенной влажности, под прямыми солнечными лучами;

– не накрывайте подключенные к сети устройства (телефон, адаптер, беспроводную зарядку) одеялами, подушками и другими предметами – это может привести к перегреву и возгоранию.

Расстояние между телом и телефоном должно быть не меньше 1 см [6].

Во избежание несчастных случаев (а также потому, что это является нарушением закона в большинстве стран) не пользуйтесь телефоном за рулем, во время управления производственным оборудованием, а также при осуществлении других потенциально опасных видов деятельности [6].

Соблюдайте правила авиалиний по использованию телефонов во время полета и перед посадкой. Использование мобильных устройств в самолете может быть запрещено законом, а также препятствовать работе беспроводных

сетей и представлять опасность для воздушных судов. Чтобы избежать этого, переведите телефон в режим полета [6].

Соблюдение этих правил обеспечит долговечность устройства, защитит от травм и сохранит вашу безопасность.

Заключение

В ходе реализации дипломной работы мною был проведён анализ нескольких свёрточных нейронных сетей, была выбрана модель с наилучшими показателями, которая впоследствии была адаптирована с помощью технологии трансферного обучения. С использованием паттерна MVVM, обеспечившего чёткое разделение логики на слои, модель была интегрирована в проект. Слой Model включает работу с базой данных SQLite для хранения метаданных изображений и классификатор на базе TensorFlow Lite. ViewModel координирует процессы синхронизации данных, а View предоставляет интуитивно понятный интерфейс.

В время выполнения данной работы было разработано Android-приложение для локальной классификации изображений с использованием свёрточной нейронной сети. Проект демонстрирует возможность эффективного внедрения технологий искусственного интеллекта в мобильные приложения. В приложении реализован следующий функционал:

– полностью автономная обработка всех изображений пользователя, что обеспечивает конфиденциальность данных;

– классификация на категории: «Авто», «Документы», «Еда», «Животные», «Люди», «Не определено»;

– просмотр итогового результата.

Дальнейшее развитие включает добавление новых категорий классификации, а также возможность дообучения модели самим пользователем, что позволит приложению с каждым разом лучше обрабатывать изображения. Например, если в папке «Авто» окажется фотография человека, пользователь сможет вручную перенести её в соответствующую категорию («Люди»). При этом приложение будет запоминать этот выбор и использовать его для последующего обучения модели, постепенно улучшая точность классификации.

Список использованных источников

1. Google Фото. [Электронный ресурс]. URL:https://www.google.com/photos/about/ (дата обращения 09.05.25)

2. Google Объектив. [Электронный ресурс]. URL: https://lens.google/ (дата обращения 10.05.25)

3. Галерея ИИ. [Электронный ресурс]. URL: https://play.google.com/store/apps/details?id=gallery.ai.photoeditor&hl=ru (дата обращения 10.04.25)

4. Smart Image Classifier. [Электронный ресурс]. URL: https://github.com/kanugurajesh/Image-Classification (дата обращения 09.05.25)

5. Трансферное обучение и тонкая настройка. [Электронный ресурс]. URL: https://www.tensorflow.org/tutorials/images/transfer_learning?hl=ru (дата обращения 16.05.25)

6. Информация о технике безопасности. [Электронный ресурс]. URL: https://support.google.com/pixelphone/answer/7022290?hl=ru (дата обращения 28.05.25)

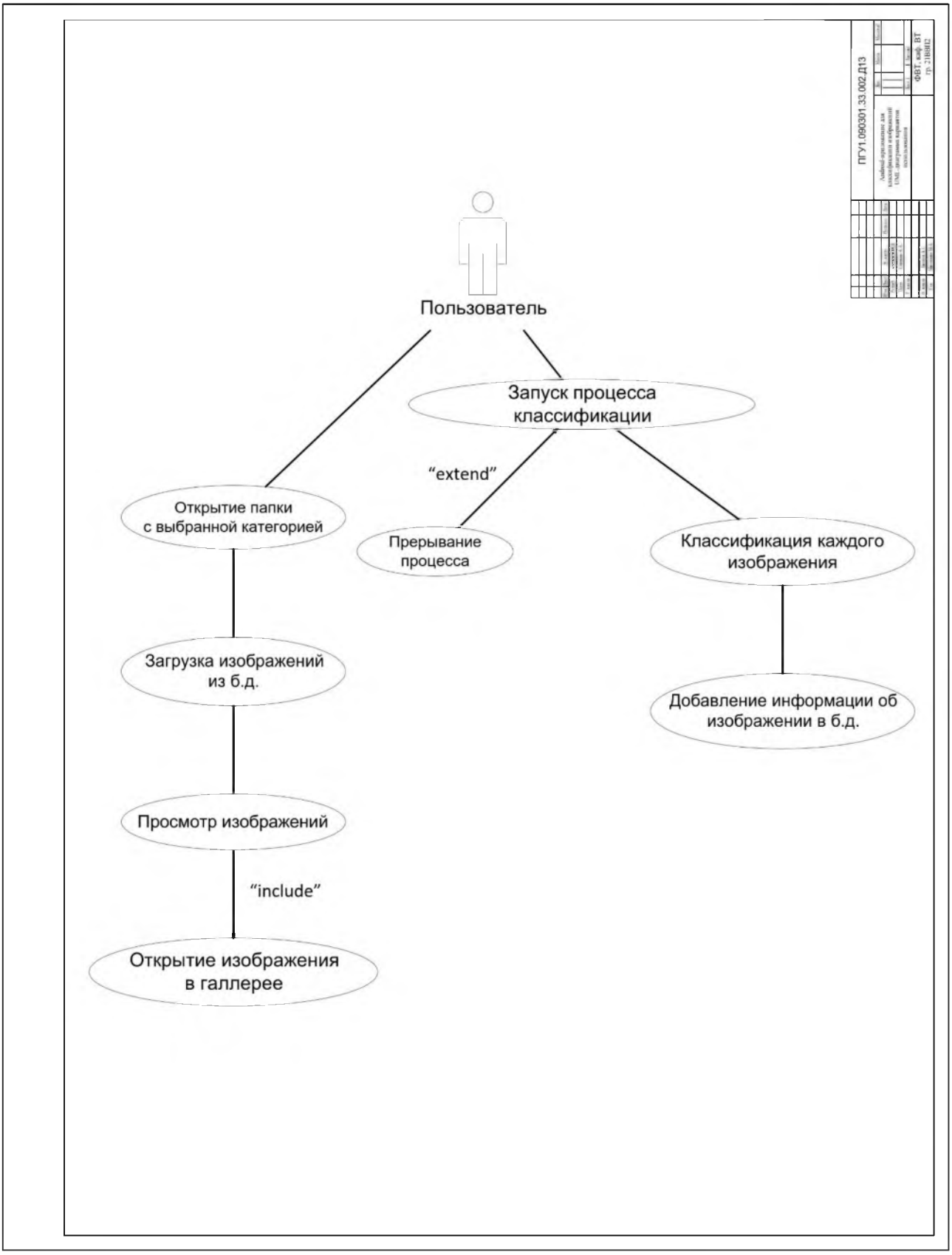
Приложение А

(Обязательное)

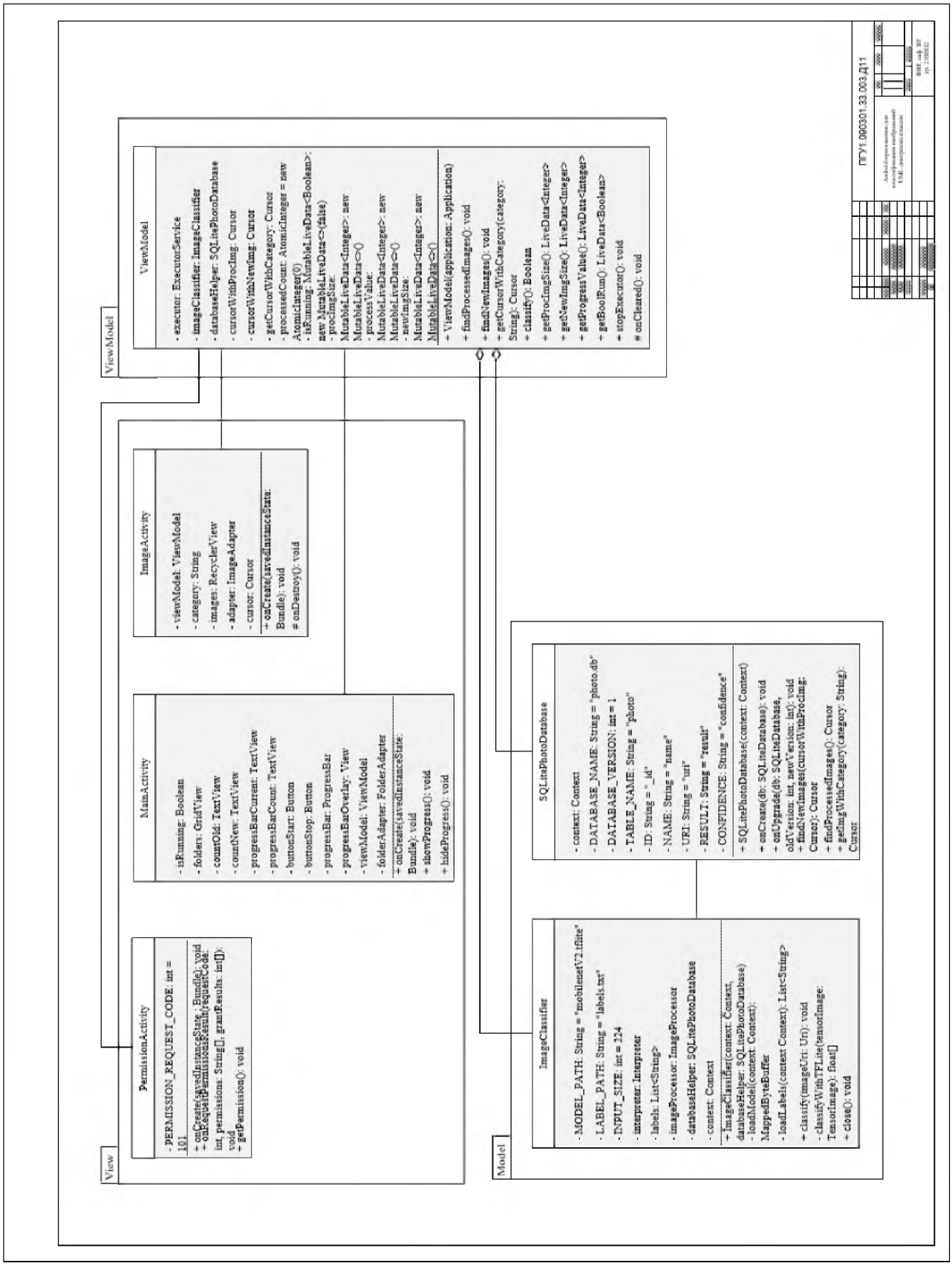
Android-приложение для классификации изображений

UML-диаграммы

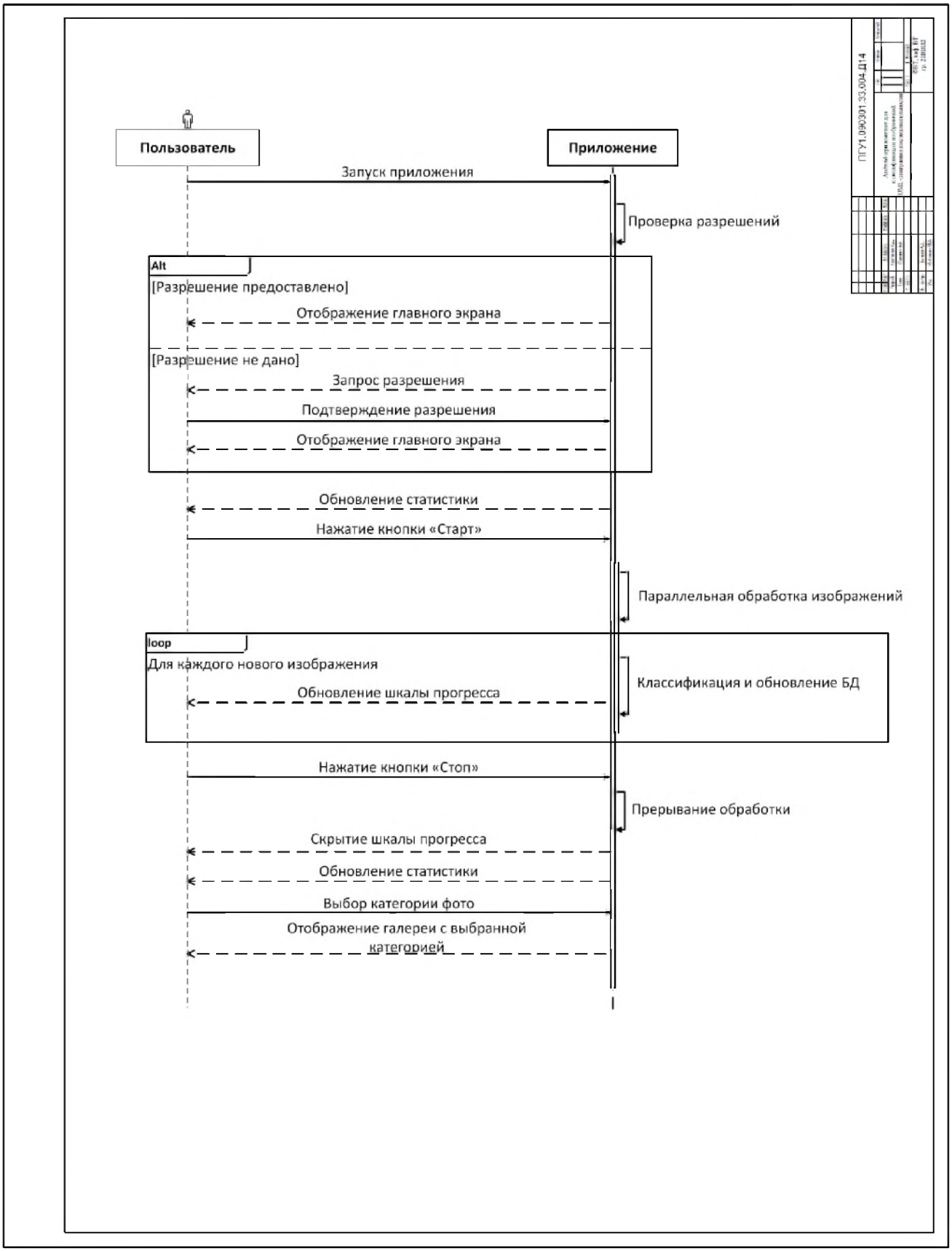
Приложение А.1 – UML-диаграмма вариантов использования



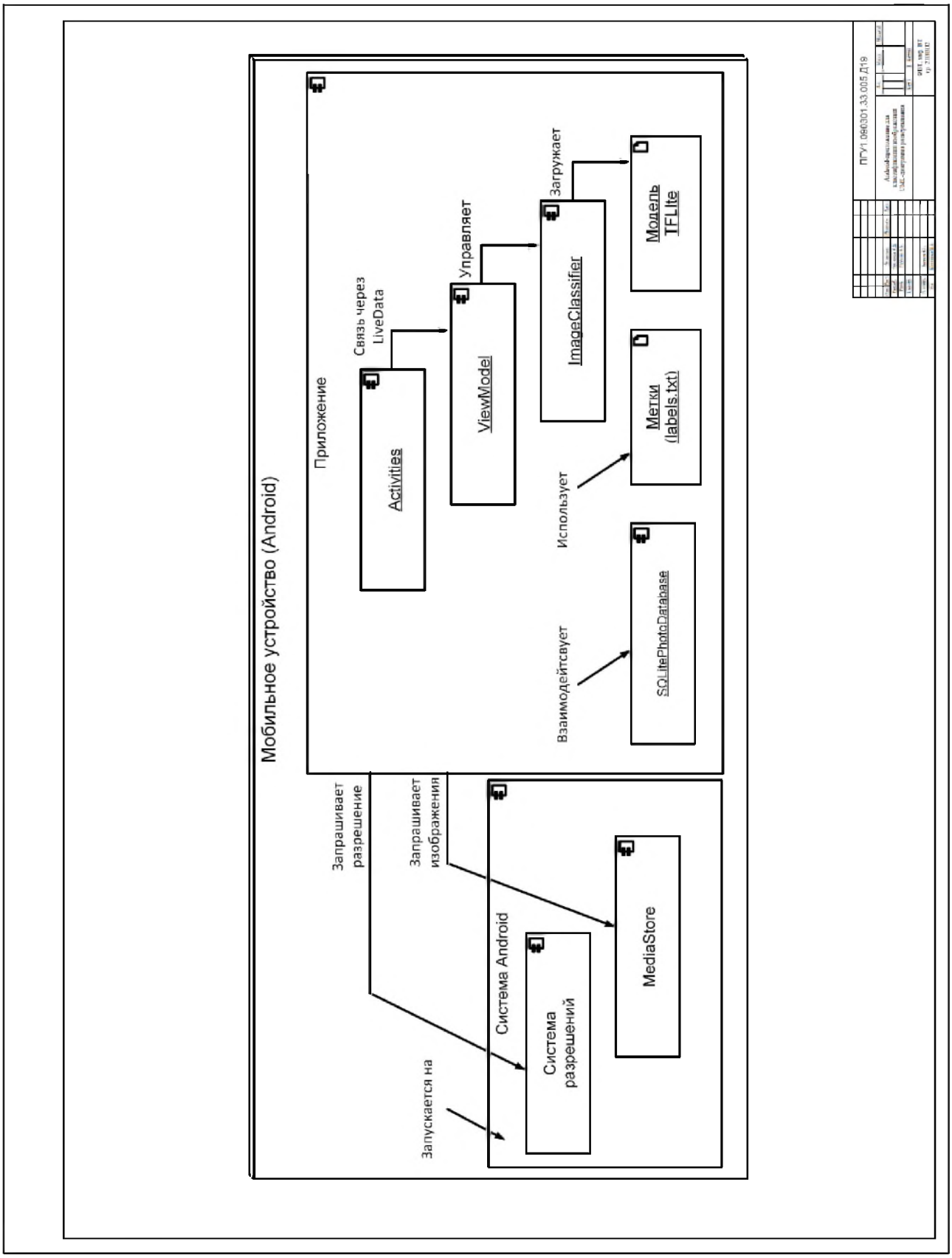
Приложение А.2 – UML-диаграмма классов



Приложение А.3 – UML-диаграмма последовательности



Приложение А.4 – UML-диаграмма развёртывания



Приложение Б

(Обязательное)

Android-приложение для классификации изображений

Листинг программы

Файл «mobilenetTransferLearning.py»

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.preprocessing import
image_dataset_from_directory
from PIL import Image
import os

#загрузка данных
BATCH_SIZE = 32
IMG_SIZE = (224, 224)
directory = "C:\\Users\\iljak\\Desktop\\test_dataset"
train_dataset = image_dataset_from_directory(directory,
                                             shuffle=True,

batch_size=BATCH_SIZE,

image_size=IMG_SIZE,

validation_split=0.2,
                                             subset='training',
                                             seed=42)
validation_dataset = image_dataset_from_directory(directory,
                                             shuffle=True,

batch_size=BATCH_SIZE,

image_size=IMG_SIZE,

validation_split=0.2,

subset='validation',
                                             seed=42)

val_batches =
tf.data.experimental.cardinality(validation_dataset)
test_dataset = validation_dataset.take(val_batches // 5)
validation_dataset = validation_dataset.skip(val_batches // 5)

#первые девять изображений и меток из обучающего набора
class_names = train_dataset.class_names
plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
plt.show()
```

```
#буферизованная предварительная выборка для загрузки изображений
AUTOTUNE = tf.data.experimental.AUTOTUNE
train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset =
validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)

#увеличение данных для уменьшения переобучения модели
data_augmentation = tf.keras.Sequential([
  tf.keras.layers.RandomFlip('horizontal'),
  tf.keras.layers.RandomRotation(0.2),
])

for image, _ in train_dataset.take(1):
  # plt.figure(figsize=(10, 10))
  first_image = image[0]
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    augmented_image =
data_augmentation(tf.expand_dims(first_image, 0))
    plt.imshow(augmented_image[0] / 255)
    plt.axis('off')
plt.show()


#метод предварительной обработки пискелей, т.к. модель ожидает
значения пикселей в [-1, 1],
#но на данный момент значения пикселей находятся в [0, 255]
preprocess_input =
tf.keras.applications.mobilenet_v2.preprocess_input

# Создание базовой модели на основе обученной модели moblinenet
IMG_SHAPE = IMG_SIZE + (3,)
base_model =
tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,

include_top=False,#удалить верхний слой

weights='imagenet')
base_model.summary()

image_batch, label_batch = next(iter(train_dataset))
feature_batch = base_model(image_batch)
print(feature_batch.shape)

global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)
```

```
prediction_layer = tf.keras.layers.Dense(5, activation =
'softmax')
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)

#заморозка слоёв базовой модели
base_model.trainable = False
#входной слой
inputs = tf.keras.Input(shape=(224, 224, 3))
x = data_augmentation(inputs)
x = preprocess_input(x)
x = base_model(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)

#скорость обучения
base_learning_rate = 0.0001
#компиляция модели
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=b
ase_learning_rate),

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=F
alse),
              metrics=['accuracy'])
model.summary()


#запуск обучения
initial_epochs = 10
loss0, accuracy0 = model.evaluate(validation_dataset)
print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))
history = model.fit(train_dataset,
                    epochs=initial_epochs,
                    validation_data=validation_dataset)

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
```

```
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,1.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()


base_model.trainable = True

# Let's take a look to see how many layers are in the base model
print("Number of layers in the base model: ",
len(base_model.layers))

# Fine-tune from this layer onwards
fine_tune_at = 100

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable = False

model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy
(from_logits=False),
              optimizer =
tf.keras.optimizers.RMSprop(learning_rate=base_learning_rate/10)
,
              metrics=['accuracy'])
len(model.trainable_variables)

fine_tune_epochs = 10
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model.fit(train_dataset,
                         epochs=total_epochs,
                         initial_epoch=history.epoch[-1],
                         validation_data=validation_dataset)


#проверка на тестовых данных
loss, accuracy = model.evaluate(test_dataset)
print('Test accuracy :', accuracy)

# Retrieve a batch of images from the test set
image_batch, label_batch =
test_dataset.as_numpy_iterator().next()
predictions = model.predict_on_batch(image_batch)
```

```
predicted_classes = tf.argmax(predictions, axis=1).numpy()
# Apply a sigmoid since our model returns logits

print('Predictions:\n', predicted_classes)
print('Labels:\n', label_batch)

model.summary()

#удаляем слои аугментации для конвертации в формат TFLite
inference_inputs = tf.keras.Input(shape=(224, 224, 3))  # Новый
входной слой
x = preprocess_input(inference_inputs)
x = base_model(x, training=False)
x = global_average_layer(x)
x = model.layers[-2](x)
inference_outputs = model.layers[-1](x)
inference_model = tf.keras.Model(inference_inputs,
inference_outputs)

inference_model.compile(

optimizer=tf.keras.optimizers.Adam(learning_rate=base_learning_r
ate),

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=F
alse),
    metrics=['accuracy']
)

# Проверяем архитектуру
inference_model.summary()

#сохранение модели
inference_model.export("C:\\Users\\iljak\\Downloads\\model")
inference_model.save("C:\\Users\\iljak\\Downloads\\model\\model.
keras")

#конвертация в TFLite
converter =
tf.lite.TFLiteConverter.from_keras_model(inference_model)
tflite_model = converter.convert()
with
open('C:\\Users\\iljak\\Downloads\\model\\mobilenet.tflite',
'wb') as f:
    f.write(tflite_model)

#проверка на своих данных
directory = "C:\\Users\\iljak\\Downloads\\random"
files = os.listdir(directory)
print(files)
```

```
def preprocess_image(image):
    image = image.resize((224, 224))
    image = np.array(image).astype(np.float32)
    image = np.expand_dims(image, axis=0)
    return image


for file in files :
     img_path = os.path.join(directory, file)
     image = Image.open(img_path)
     preprocessed = preprocess_image(image)
     outputs = inference_model(preprocessed)
     print(outputs)
     predictions = tf.where(outputs < 0.5, 0, 1)
     print(predictions)
     print("Model output:",
class_names[predictions.numpy().argmax()])
     fig = plt.figure(figsize=(10, 10))
     ax = fig.add_subplot()
     plt.title(class_names[predictions.numpy().argmax()])
     ax.imshow(image)
     plt.show()
```

Файл «ImageClassifier.java»

```
package com.example.mobilenetv2imageclassification.model;

import android.content.ContentValues;
import android.content.Context;
import android.content.res.AssetFileDescriptor;
import android.database.sqlite.SQLiteDatabase;
import android.graphics.Bitmap;
import android.net.Uri;
import android.provider.MediaStore;
import android.util.Log;
import org.tensorflow.lite.DataType;
import org.tensorflow.lite.Interpreter;
import org.tensorflow.lite.support.image.ImageProcessor;
import org.tensorflow.lite.support.image.ops.ResizeOp;
import org.tensorflow.lite.support.image.TensorImage;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.nio.FloatBuffer;
import java.nio.MappedByteBuffer;
import java.nio.channels.FileChannel;
import java.util.ArrayList;
import java.util.List;

public class ImageClassifier {
```

```
    private static final String MODEL_PATH =
"mobilenetV2.tflite";
    private static final String LABEL_PATH = "labels.txt";
    private static final int INPUT_SIZE = 224;
    private final Interpreter interpreter;
    private final List<String> labels;
    private final ImageProcessor imageProcessor;
    SQLitePhotoDatabase databaseHelper;
    Context context;

    public ImageClassifier(Context context, SQLitePhotoDatabase
databaseHelper) throws IOException {
        this.context = context;
        interpreter = new Interpreter(loadModel(context));
        labels = loadLabels(context);
        imageProcessor = new ImageProcessor.Builder()
                .add(new ResizeOp(INPUT_SIZE, INPUT_SIZE,
ResizeOp.ResizeMethod.BILINEAR))
                .build();
        this.databaseHelper = databaseHelper;
    }

    private MappedByteBuffer loadModel(Context context) throws
IOException {
        AssetFileDescriptor fileDescriptor =
context.getAssets().openFd(MODEL_PATH);
        FileInputStream inputStream = new
FileInputStream(fileDescriptor.getFileDescriptor());
        FileChannel fileChannel = inputStream.getChannel();
        long startOffset = fileDescriptor.getStartOffset();
        long declaredLength =
fileDescriptor.getDeclaredLength();
        return fileChannel.map(FileChannel.MapMode.READ_ONLY,
startOffset, declaredLength);
    }

    private List<String> loadLabels(Context context) throws
IOException {
        List<String> labelList = new ArrayList<>();
        BufferedReader reader = new BufferedReader(new
InputStreamReader(context.getAssets().open(LABEL_PATH)));
        String line;
        while ((line = reader.readLine()) != null) {
            labelList.add(line);
        }
        reader.close();
        return labelList;
    }

    public void classify(Uri imageUri){
        Bitmap bitmap = null;
```

```
        try {
            bitmap =
MediaStore.Images.Media.getBitmap(context.getContentResolver(),
imageUri);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
        Bitmap image = bitmap.copy(Bitmap.Config.ARGB_8888,
true);
        TensorImage tensor = new TensorImage(DataType.FLOAT32);
        tensor.load(image);
        TensorImage processedTensor =
imageProcessor.process(tensor);
        float[] output = classifyWithTFLite(processedTensor);
        int maxIndex = 0;
        float maxConfidence = 0;
        for (int i = 0; i < output.length; i++) {
            if (output[i] > maxConfidence) {
                maxConfidence = output[i];
                maxIndex = i;
            }
        }
        SQLiteDatabase db =
databaseHelper.getWritableDatabase();
        ContentValues contentValues = new ContentValues();
        if (maxConfidence < 0.95) {
            contentValues.put("result", "Не определено");
        }
        else {
            contentValues.put("result", labels.get(maxIndex));
        }
        contentValues.put("confidence", "(" + (maxConfidence *
100) + "%)");
        db.update("photo", contentValues, "uri = \"" + imageUri
+ "\";", null);
        Log.d("Classification", "Label: " + labels.get(maxIndex)
+ ", Confidence: " + maxConfidence);
    }

    private float[] classifyWithTFLite(TensorImage tensorImage)
{
        int[] outputShape =
interpreter.getOutputTensor(0).shape();
        FloatBuffer outputBuffer =
FloatBuffer.allocate(outputShape[1]);
        outputBuffer.rewind();

interpreter.run(tensorImage.getTensorBuffer().getBuffer(),
outputBuffer);
        outputBuffer.rewind();
        float[] output = new float[outputBuffer.capacity()];
```

```
        outputBuffer.get(output);
        return output;
    }

    public void close(){
        interpreter.close();
    }
}
```

Файл «SQLitePhotoDatabase.java»

```
package com.example.mobilenetv2imageclassification.model;

import android.content.ContentUris;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.MatrixCursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.net.Uri;
import android.provider.MediaStore;
import androidx.annotation.Nullable;
import java.io.IOException;
import java.util.HashSet;
import java.util.Set;

public class SQLitePhotoDatabase extends SQLiteOpenHelper {

    private final Context context;
    private static final String DATABASE_NAME = "photo.db";
    private static final int DATABASE_VERSION = 1;
    private static final String TABLE_NAME = "photo";
    private static final String ID = "_id";
    private static final String NAME = "name";
    private static final String URI = "uri";
    private static final String RESULT = "result";
    private static final String CONFIDENCE = "confidence";

    public SQLitePhotoDatabase(@Nullable Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
        this.context = context;
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        String query = "CREATE TABLE " + TABLE_NAME + " ("
                + ID + " INTEGER PRIMARY KEY, "
                + NAME + " TEXT, " + URI + " TEXT, " + RESULT +
" TEXT, " + CONFIDENCE + " TEXT);";
        db.execSQL(query);
```

```
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
        onCreate(db);
    }

    /**
     * Метод сравнивает два курсора: один, полученный из б.д.
SQLite, второй - из MediaStore.
     * Сравнение происходит путём сравнивания ID изображения из
SQLite с ID из MediaStore.
     * Если совпадений нет, то запись о новом изображении
попадает в результирующий курсор.
     * @param cursorWithProcImg Курсор с данными из SQLite
     * @return Cursor с данными для последующей обработки
     * @throws IOException
     */
    public Cursor findNewImages(Cursor cursorWithProcImg) {
        Set<Long> existingIds = new HashSet<>();
        if (cursorWithProcImg != null &&
cursorWithProcImg.moveToFirst()) {
            do {
                long id =
cursorWithProcImg.getLong(cursorWithProcImg.getColumnIndexOrThro
w("_id"));
                existingIds.add(id);
            } while (cursorWithProcImg.moveToNext());
        }
        String[] projection = {
                MediaStore.Images.Media._ID,
                MediaStore.Images.Media.DISPLAY_NAME,
        };

        Cursor mediaStoreCursor =
context.getContentResolver().query(
                MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
                projection,
                null,
                null,
                MediaStore.Images.Media.DATE_TAKEN + " DESC"
        );
        MatrixCursor resultCursor = new MatrixCursor(new
String[]{
                MediaStore.Images.Media._ID,
                MediaStore.Images.Media.DISPLAY_NAME,
                "uri"
        });
        if (mediaStoreCursor != null) {
```

```
            try {
                while (mediaStoreCursor.moveToNext()) {
                    long id =
mediaStoreCursor.getLong(mediaStoreCursor.getColumnIndexOrThrow(
MediaStore.Images.Media._ID));
                    if (!existingIds.contains(id)) {
                        String name =
mediaStoreCursor.getString(mediaStoreCursor.getColumnIndexOrThro
w(MediaStore.Images.Media.DISPLAY_NAME));
                        Uri imageUri =
ContentUris.withAppendedId(MediaStore.Images.Media.EXTERNAL_CONT
ENT_URI, id);
                        SQLiteDatabase dbWriter =
this.getWritableDatabase();
                        ContentValues values = new
ContentValues();
                        values.put("_id", id);
                        values.put("name", name);
                        values.put("uri", imageUri.toString());
                        dbWriter.insertWithOnConflict("photo",
null, values, SQLiteDatabase.CONFLICT_REPLACE);
                        resultCursor.addRow(new Object[]{id,
name, imageUri.toString()});
                    }
                }
            } finally {
                mediaStoreCursor.close();
            }
        }
        return resultCursor;
    }

    /**
     *
     * @return Cursor с данными из б.д., где есть результат
классификаиии. Нужен для сравнения.
     */
    public Cursor findProcessedImages(){
        SQLiteDatabase dbReader = this.getReadableDatabase();
        String selection = "result IS NOT NULL";
        return dbReader.query(
                "photo", null, selection,null,null,null,null
        );
    }

    public Cursor getImgWithCategory(String category){
        SQLiteDatabase dbReader = this.getReadableDatabase();
        String selection = null;
        switch (category){
            case "Авто": selection = "result IS 'vehicle'";
break;
```

```
            case "Документы": selection = "result IS
'document'"; break;
            case "Еда": selection = "result IS 'food'"; break;
            case "Животные": selection = "result IS 'animals'";
break;
            case "Люди": selection = "result IS 'human'"; break;
            case "Не определено": selection = "result IS 'Не
определено'"; break;
        }
        return dbReader.query(
                "photo", null, selection,null,null,null,null
        );
    }
}
```

Файл «ViewModel.java»

```
package com.example.mobilenetv2imageclassification.viewModel;

import android.app.Application;
import android.content.ContentUris;
import android.database.Cursor;
import android.os.Handler;
import android.os.Looper;
import android.provider.MediaStore;
import android.util.Log;
import androidx.annotation.NonNull;
import androidx.lifecycle.AndroidViewModel;
import androidx.lifecycle.LiveData;
import androidx.lifecycle.MutableLiveData;
import
com.example.mobilenetv2imageclassification.model.ImageClassifier
;
import
com.example.mobilenetv2imageclassification.model.SQLitePhotoData
base;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.atomic.AtomicInteger;

public class ViewModel extends AndroidViewModel {
    private ExecutorService executor;
    private final ImageClassifier imageClassifier;
    private final SQLitePhotoDatabase databaseHelper;
    private Cursor cursorWithProcImg;
    private Cursor cursorWithNewImg;
    private Cursor getCursorWithCategory;
    AtomicInteger processedCount = new AtomicInteger(0);
```

```
    private final MutableLiveData<Boolean> isRunning = new
MutableLiveData<>(false);
    private final MutableLiveData<Integer> procImgSize = new
MutableLiveData<>();
    private final MutableLiveData<Integer> newImgSize = new
MutableLiveData<>();
    private final MutableLiveData<Integer> processValue = new
MutableLiveData<>();

    public ViewModel(@NonNull Application application) {
        super(application);
        executor =
Executors.newFixedThreadPool(Runtime.getRuntime().availableProce
ssors() / 2);
        databaseHelper = new
SQLitePhotoDatabase(ViewModel.this.getApplication().getApplicati
onContext());
        try {
            imageClassifier = new
ImageClassifier(ViewModel.this.getApplication().getApplicationCo
ntext(), databaseHelper);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    public void findProcessedImages() {
        if (cursorWithProcImg != null) {
            cursorWithProcImg.close();
        }
        cursorWithProcImg =
databaseHelper.findProcessedImages();
        procImgSize.postValue(cursorWithProcImg.getCount());
    }

    public void findNewImages() {
        if (cursorWithNewImg != null) {
            cursorWithNewImg.close();
        }
        cursorWithNewImg =
databaseHelper.findNewImages(cursorWithProcImg);
        newImgSize.postValue(cursorWithNewImg.getCount());
    }

    public Cursor getCursorWithCategory(String category){
        if (getCursorWithCategory != null) {
            getCursorWithCategory.close();
        }
        return databaseHelper.getImgWithCategory(category);
    }
```

```
    public Boolean classify() {
        if (cursorWithNewImg == null ||
cursorWithNewImg.getCount() == 0) {
            return false;
        }
        isRunning.postValue(true);
        processedCount.set(0);
        if (executor.isShutdown()){
            executor =
Executors.newFixedThreadPool(Runtime.getRuntime().availableProce
ssors() / 2);
        }
        List<Long> imageIds = new ArrayList<>();
        cursorWithNewImg.moveToFirst();
        do {
            long id =
cursorWithNewImg.getLong(cursorWithNewImg.getColumnIndexOrThrow(
MediaStore.Images.Media._ID));
            imageIds.add(id);
        } while (cursorWithNewImg.moveToNext());
        for (Long id : imageIds) {
            executor.execute(() -> {
                try {

imageClassifier.classify(ContentUris.withAppendedId(MediaStore.I
mages.Media.EXTERNAL_CONTENT_URI, id));
                } catch (Exception ex){
                    Log.e("Thread", ex.getMessage());
                } finally {
                    int currentValue =
processedCount.incrementAndGet();
                    processValue.postValue(currentValue);
                    if (currentValue == imageIds.size()) {
                        new
Handler(Looper.getMainLooper()).post(() -> {
                            findProcessedImages();
                            findNewImages();
                            isRunning.postValue(false);
                        });
                    }
                }
            });
        }
        return true;
    }

    public LiveData<Integer> getProcImgSize() {
        return procImgSize;
    }

    public LiveData<Integer> getNewImgSize() {
```

```
        return newImgSize;
    }

    public LiveData<Integer> getProgressValue(){
        return processValue;
    }

    public LiveData<Boolean> getBoolRun(){
        return isRunning;
    }

    public void stopExecutor(){
        findProcessedImages();
        findNewImages();
        isRunning.postValue(false);
        executor.shutdownNow();
    }

    @Override
    protected void onCleared(){
        executor.shutdownNow();
        imageClassifier.close();
        databaseHelper.close();
        if (cursorWithProcImg != null) {
            cursorWithProcImg.close();
        }
        if (cursorWithNewImg != null) {
            cursorWithNewImg.close();
        }
    }
}
```

Файл «PermissionActivity.java»

```
package com.example.mobilenetv2imageclassification.view;

import android.Manifest;
import android.app.Activity;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.os.Build;
import android.os.Bundle;
import android.widget.Toast;
import androidx.annotation.NonNull;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;

public class PermissionActivity extends Activity {
    private static final int PERMISSION_REQUEST_CODE = 101;
    @Override
```

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        getPermission();
    }
    @Override
    public void onRequestPermissionsResult(int requestCode,
@NonNull String[] permissions, @NonNull int[] grantResults){
        if (requestCode == PERMISSION_REQUEST_CODE) {
            if (grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
                startActivity(new Intent(this,
MainActivity.class));
                finish();
            } else {
                Toast.makeText(PermissionActivity.this, "Без
данного разрешения приложение не сможет обрабатывать ваши
изображения, пожалуйста, дайте разрешение.",
Toast.LENGTH_SHORT).show();
                getPermission();
            }
        }
    }
    public void getPermission(){
        String permission;
        if (Build.VERSION.SDK_INT >=
Build.VERSION_CODES.TIRAMISU) {
            permission =
android.Manifest.permission.READ_MEDIA_IMAGES;
        } else {
            permission =
Manifest.permission.READ_EXTERNAL_STORAGE;
        }
        if (ContextCompat.checkSelfPermission(this, permission)
                != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(
                    this,
                    new String[]{permission},
                    PERMISSION_REQUEST_CODE
            );
        }
        else {
            startActivity(new Intent(this, MainActivity.class));
            finish();
        }
    }
}
```

Файл «MainActivity.java»

```
package com.example.mobilenetv2imageclassification.view;
```

```
import android.annotation.SuppressLint;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.Button;
import android.widget.GridView;
import android.widget.ProgressBar;
import android.widget.TextView;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import androidx.lifecycle.ViewModelProvider;
import com.example.mobilenetv2imageclassification.R;
import
com.example.mobilenetv2imageclassification.adapters.FolderAdapte
r;
import
com.example.mobilenetv2imageclassification.viewModel.ViewModel;

public class MainActivity extends AppCompatActivity {

    Boolean isRunning;
    GridView folders;
    TextView countOld;
    TextView countNew;
    TextView progressBarCurrent;
    TextView progressBarCount;
    Button buttonStart;
    Button buttonStop;
    ProgressBar progressBar;
    View progressBarOverlay;
    ViewModel viewModel;
    FolderAdapter folderAdapter;

    @SuppressLint("MissingInflatedId")
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        folders = findViewById(R.id.folders);
        countOld = findViewById(R.id.countImages);
        countNew = findViewById(R.id.countNewImage);
        progressBarCurrent =
findViewById(R.id.progressBarCurrent);
        progressBarCount = findViewById(R.id.progressBarCount);
        buttonStart = findViewById(R.id.buttonStart);
        buttonStop = findViewById(R.id.closeProgressBar);
        progressBar = findViewById(R.id.progressBar);
        progressBarOverlay = findViewById(R.id.progressOverlay);
        viewModel = new
ViewModelProvider(this).get(ViewModel.class);
```

```
        folderAdapter = new FolderAdapter(this);
        folders.setAdapter(folderAdapter);

        viewModel.findProcessedImages();
        viewModel.findNewImages();

        viewModel.getProcImgSize().observe(this, result -> {
            countOld.setText("Количество обработанных
изображений = " + String.valueOf(result));
        });

        viewModel.getNewImgSize().observe(this, result -> {
            countNew.setText("Количество новых изображений = " +
String.valueOf(result));
            progressBar.setMax(result);
            progressBarCount.setText(String.valueOf(result));
        });

        viewModel.getProgressValue().observe(this, result -> {
            progressBar.setProgress(result);
            progressBarCurrent.setText(String.valueOf(result));
        });

        viewModel.getBoolRun().observe(this, result -> {
            isRunning = result;
            if (!isRunning) {
                hideProgress();
            }
        });

        folders.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View
view, int position, long id) {
                String imgClass =
parent.getItemAtPosition(position).toString();
                Intent intent = new
Intent(getApplicationContext(), ImageActivity.class);
                intent.putExtra("class", imgClass);
                startActivity(intent);
            }
        });

        buttonStart.setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (!isRunning) {
                    if (viewModel.classify()){
                       showProgress();
```

```
                    }
                    else {
                        Toast toast =
Toast.makeText(getApplicationContext(), "Нет новых изображений
для обработки", Toast.LENGTH_LONG);
                        toast.show();
                    }
                }
            }
        });

        buttonStop.setOnClickListener(new View.OnClickListener()
{
            @Override
            public void onClick(View v) {
                hideProgress();
            }
        });
    }

    void showProgress(){
        progressBarOverlay.setVisibility(View.VISIBLE);
        progressBar.setProgress(0);
        progressBarCurrent.setText("0");
    }

    void hideProgress(){
        progressBarOverlay.setVisibility(View.GONE);
        viewModel.stopExecutor();
    }
}
```

Файл «ImageActivity.java»

```
package com.example.mobilenetv2imageclassification.view;

import android.database.Cursor;
import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;
import androidx.lifecycle.ViewModelProvider;
import androidx.recyclerview.widget.RecyclerView;
import androidx.recyclerview.widget.StaggeredGridLayoutManager;
import com.example.mobilenetv2imageclassification.R;
import
com.example.mobilenetv2imageclassification.adapters.ImageAdapter
;
import
com.example.mobilenetv2imageclassification.viewModel.ViewModel;

public class ImageActivity extends AppCompatActivity {
    ViewModel viewModel;
```

```
    String category;
    RecyclerView images;
    ImageAdapter adapter;
    Cursor cursor;
    @Override
    public void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.image_activity);
        viewModel = new
ViewModelProvider(this).get(ViewModel.class);
        images = findViewById(R.id.images);
        images.setLayoutManager(new
StaggeredGridLayoutManager(2,StaggeredGridLayoutManager.VERTICAL
));
        images.setHasFixedSize(true);
        Bundle arguments = getIntent().getExtras();
        category = arguments.get("class").toString();
        cursor = viewModel.getCursorWithCategory(category);
        adapter = new ImageAdapter(cursor,
getApplicationContext());
        images.setAdapter(adapter);

    }

    @Override
    protected void onDestroy() {
        if (cursor != null && !cursor.isClosed()) {
            cursor.close();
        }
        super.onDestroy();
    }
}
```

Файл « FolderAdapter.java»

```
package com.example.mobilenetv2imageclassification.adapters;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.TextView;
import com.example.mobilenetv2imageclassification.R;
import java.util.Arrays;
import java.util.List;

public class FolderAdapter extends BaseAdapter {
```

```
    private final List<String> categories =
Arrays.asList("Авто", "Еда", "Документы", "Животные", "Люди",
"Не определено");
    LayoutInflater inflater;

    public FolderAdapter(Context context) {
        inflater = LayoutInflater.from(context);
    }

    @Override
    public int getCount() {
        return categories.size();
    }

    @Override
    public Object getItem(int position) {
        return categories.get(position);
    }

    @Override
    public long getItemId(int position) {
        return 0;
    }

    @Override
    public View getView(int position, View convertView,
ViewGroup parent) {
        ViewHolder holder;
        if (convertView == null) {
            convertView =
inflater.inflate(R.layout.folders_layout, parent, false);
            holder = new ViewHolder();
            holder.folderName =
convertView.findViewById(R.id.folderName);
            convertView.setTag(holder);
        } else {
            holder = (ViewHolder) convertView.getTag();
        }
        holder.folderName.setText(categories.get(position));
        return convertView;
    }

    static class ViewHolder {
        TextView folderName;
    }
}
```

Файл «ImageAdapter.java»

```
package com.example.mobilenetv2imageclassification.adapters;
```

```
import android.content.Context;
import android.content.Intent;
import android.database.Cursor;
import android.net.Uri;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;
import com.bumptech.glide.Glide;
import com.example.mobilenetv2imageclassification.R;

public class ImageAdapter extends
RecyclerView.Adapter<ImageAdapter.ViewHolder> {

    private Cursor cursor;
    private Context context;

    public class ViewHolder extends RecyclerView.ViewHolder{
        ImageView image;

        public ViewHolder(@NonNull View itemView) {
            super(itemView);
            image = itemView.findViewById(R.id.image);
        }
    }

    public ImageAdapter(Cursor cursor, Context context){
        this.cursor = cursor;
        this.context = context;
    }

    @NonNull
    @Override
    public ViewHolder onCreateViewHolder(@NonNull ViewGroup
parent, int viewType) {
        View itemView =
LayoutInflater.from(parent.getContext()).inflate(R.layout.image_
layout,parent,false);
        return new ViewHolder(itemView);
    }

    @Override
    public void onBindViewHolder(@NonNull ViewHolder holder, int
position) {
        cursor.moveToPosition(position);
        Uri uri =
Uri.parse(cursor.getString(cursor.getColumnIndexOrThrow("uri")))
;
```

```
            Glide.with(holder.itemView.getContext())
                    .load(uri)
                    .override(500,500)
                    .centerCrop()
                    .into(holder.image);

            holder.itemView.setOnClickListener(v -> {
                Intent intent = new Intent(Intent.ACTION_VIEW);
                intent.setDataAndType(uri, "image/*");
                intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                context.startActivity(intent);
            });
        }

        @Override
        public int getItemCount() {
            return cursor.getCount();
        }


    }
```

Приложение В

(Обязательное)

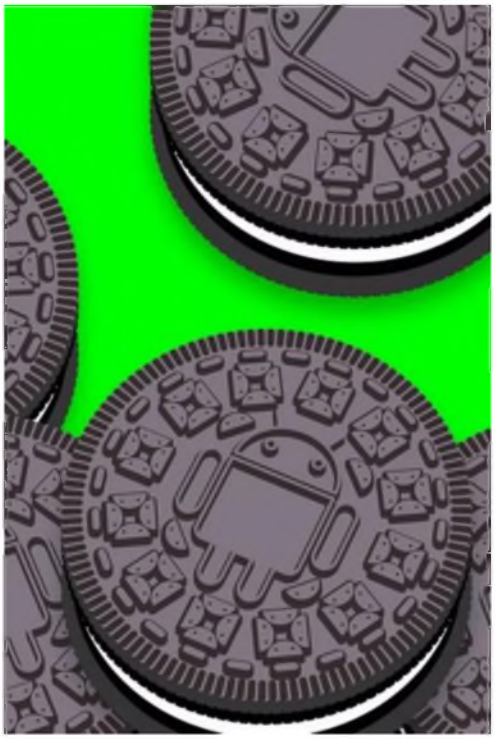
Android-приложение для классификации изображений

Презентация

Android- приложение для классификации изображений



Автор работы: студент группы 21ВВП2
Костюков Илья Дмитриевич
Научный руководитель: доцент кафедры ВТ
Сорокин Алексей Александрович



Цель работы

Разработка приложения
Локальная классификация изображений на Android

Использование CNN
Свёрточная нейронная сеть для анализа

Автономность
Работа без подключения к интернету



Актуальность

Рост мощностей

Смартфоны становятся мощнее для ИИ-задач

Популярность Android

Большое число пользователей платформы

Спрос на автономность

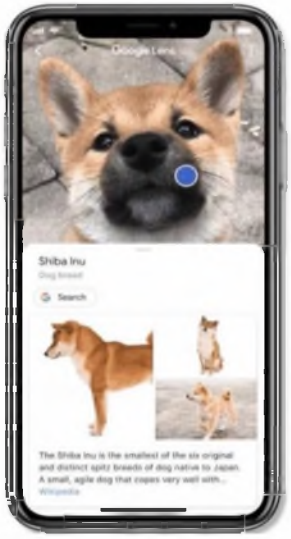
Необходимость работы ИИ без интернета

Функциональные требования

Основные функции

- Работа без интернета
- Обработка изображений на устройстве
- Классификация по 6 категориям
- Интуитивный интерфейс



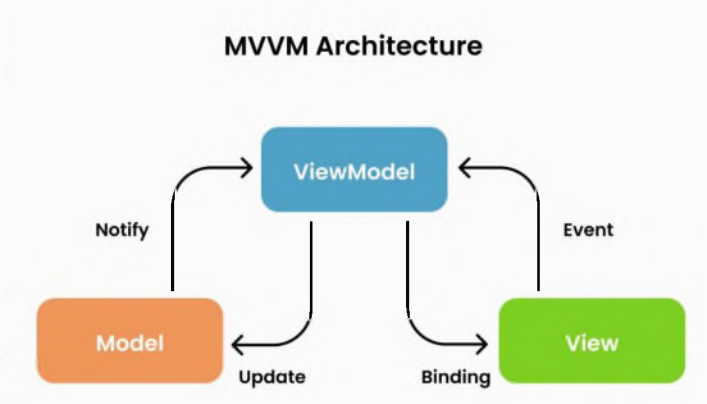


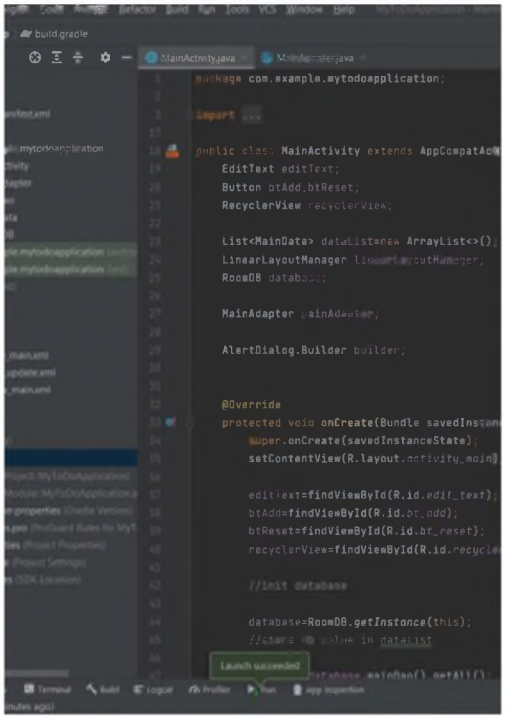
Обзор аналогов

Google Фото	Требуется интернет
Google Объектив	Классификация одного изображения
AI Gallery	Офлайн, но с рекламой
Smart Image Classifier	Офлайн, одно изображение

Выбор архитектуры

- MVVM паттерн
  Разделение логики и UI
- Model
  Классификация и база данных
- View
  Пользовательский интерфейс и отображение
- ViewModel
  Связь между UI и логикой





Технологии

Java

Основной язык разработки

Android Studio

Среда разработки

SQLite

Локальная база данных

TensorFlow Lite

Модель свёрточной нейронной сети

Обучение модели

MobileNetV2

Использована для классификации

1

2

Трансферное обучение

Ускоряет обучение модели

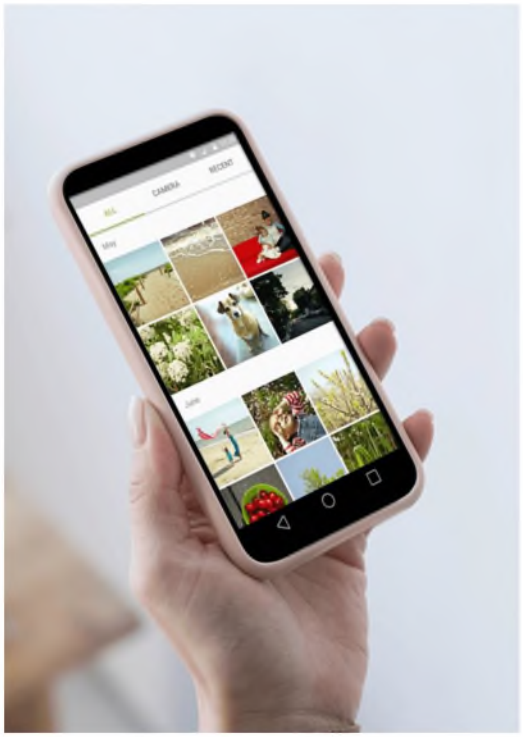
3

Экспорт в TFLite

Модель готова для мобильных устройств

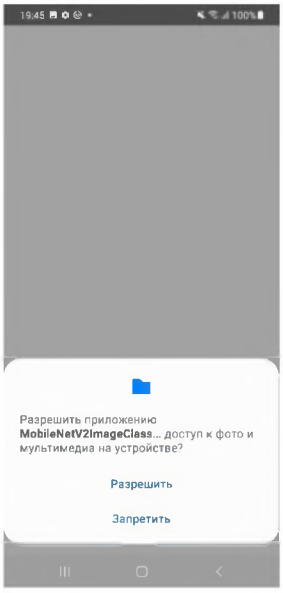
Реализация приложения

- Сканирование галереи
  Автоматический поиск изображений
- Многопоточная классификация
  Сохранение результатов в базу данных
- Просмотр изображений
  Отображение по категориям



Демонстрация функционала

- Запрос разрешения на доступ к файлам
- Автоматическое сканирование изображений



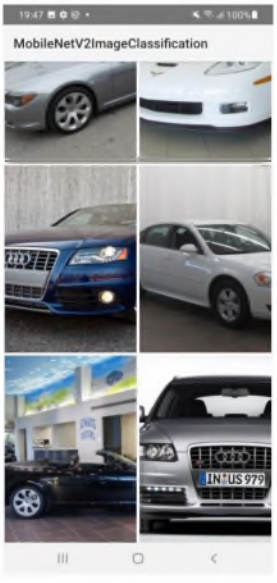
Демонстрация функционала

- Запуск классификации изображений
  - Прерывание классификации



Демонстрация функционала

- Просмотр изображений по категориям
  - Открытие изображения в галерее



Спасибо за внимание!