МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

Казахский национальный исследовательский технический университет
имени К.И. Сатпаева

Институт автоматики и информационных технологий

Кафедра «Программная инженерия»

Искакбаев Ильяс Асетович

Разработка программы для автоматизированной оценки численности тюленей на аэрофотоснимках

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту

Специальность 6B06102 – Computer Science

Алматы 2025

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

Казахский национальный исследовательский технический университет имени К.И. Сатпаева

Институт автоматики и информационных технологий

Кафедра «Программная инженерия»

ДОПУЩЕН К ЗАЩИТЕ
Заведующий кафедрой ПИ
канд. тех. наук, ассоц. профессор
_____ Ф.Н. Абдолдина
“____” _____ 2025 г.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к дипломному проекту

На тему: «Разработка программы для автоматизированной оценки численности тюленей на аэрофотоснимках»

по специальности 6B06102 - Computer Science

Выполнил Искакбаев И.А.

Рецензент
Доктор PhD, ассоциированный профессор. Заведующий кафедрой "Вычислительных наук и статистики" КазНУ им. аль-Фараби
_____ А.Н.Темирбеков
"30" 05 2025 г.

Научный руководитель
канд. техн. наук, ассоциированный профессор. Заведующий кафедрой ПИ
_____ Ф.Н.Абдолдина
"30" 05 2025 г.

Алматы 2025

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

Казахский национальный исследовательский технический университет имени К.И.Сатпаева

Институт автоматики и информационных технологий

Кафедра «Программная инженерия»

6B06102 - Computer Science

УТВЕРЖДАЮ
Заведующий кафедрой ПИ
канд. тех. наук, ассоц. профессор
_____ Ф.Н. Абдолдина
“ ____ ” _____ 2025 г.

ЗАДАНИЕ
на выполнение дипломного проекта

Обучающемуся: *Искакбаеву Ильясу Асетовичу*
Тема: Разработка программы для автоматизированной оценки численности тюленей на аэрофотоснимках

_____
Утверждена приказом проректора по академической работе: № 548-П/Ө

от «04» декабря 2024 г.

Срок сдачи законченной работы "30" мая 2025 г.

Исходные данные к дипломному проекту:
А) Анализ предметной области, анализ аналогичных проектов;
Б) Разработка технического задания;
В) Разработка архитектуры ПО;
Г) Проектирование и разработка понятного и интерактивного дизайна;
Д) Разработка и тестирование ПО;
Перечень подлежащих разработке в дипломном проекте вопросов: (с точным указанием обязательных чертежей): *представлены 17 слайдов презентации работы.*
Рекомендуемая основная литература: *из 25 наименований.*

ГРАФИК

подготовки дипломного проекта

Наименование разделов, перечень разрабатываемых вопросов	Сроки представления научному руководителю	Примечание
1. Анализ предметной области, разработка технического задания	04.12.2024	Выполнено
2. Выбор технологий для разработки	05.04.2024	Выполнено
3. Разработка логики взаимодействия	05.04.2024	Выполнено
4. Разработка функционала системы	05.04.2024	Выполнено
5. Тестирование и оптимизация	12.05.2025	Выполнено
6. Написание пояснительной записки к дипломному проекту	12.05.2025	Выполнено

Подписи

консультантов и нормоконтролера на законченный дипломный проект
с указанием относящихся к ним разделов проекта

Наименования разделов	Консультанты, И.О.Ф. (уч. степень, звание)	Дата подписания	Подпись	Оценка
Программное обеспечение	Шаханов Ә. Р., Преподаватель, магистр	30.05.25		85
Нормоконтролер	Ердалиева З.У., Преподаватель, магистр	27.05.2025		

Научный руководитель _____ Абдолдина.Ф.Н.

Задание принял к исполнению обучающийся _____ Искакбаев И.

Дата «30» 05 2025 г.

Алматы 2025

АҢДАТПА

Бұл дипломдық жоба компьютерлік көру технологияларын пайдалана отырып, Каспий итбалықтарының популяциясын бақылауға арналған мобильді қосымшаны әзірлеуге арналған. Каспий итбалықтары – сирек кездесетін және осал түрлердің бірі, олардың саны экологиялық факторлардың әсерінен азайып барады. Дәстүрлі бақылау әдістері қол жетімділіктің қиындығы мен еңбек шығындарының жоғары болуына байланысты бірнеше шектеулерге ие.

Ұсынылған шешім итбалықтардың мекендейтін аумақтарын түсіру үшін дрондарды пайдалану және кескіндерден дараларды автоматты түрде тану мен санау үшін нейрондық желі моделін қолдануды қамтиды. Қосымша клиент-серверлік архитектураға негізделген, мұнда кескіндерді талдау мобильді құрылғыда TensorFlow Lite көмегімен орындалады, ал деректер жергілікті SQLite дерекқорында сақталады.

Жұмыста нысандарды танудың заманауи әдістері, соның ішінде bounding boxes (шекаралық жақтаулар) және жылулық карталар, сондай-ақ деректерді визуализациялау интерфейстері қарастырылады. Дайындалған ақпараттық жүйе пайдаланушыларға итбалықтар популяциясы туралы мәліметтерді жинау және талдау үшін ыңғайлы құрал ұсынады.

Қосымша қарапайым және интуитивті интерфейске, кескіндерді жүктеу мүмкіндігіне, талдау нәтижелерін визуализациялауға және деректерді экспорттауға ие. Сондай-ақ, мобильді құрылғылардың шектеулі есептеу ресурстарында тиімді жұмыс істеуі үшін модельді оңтайландыру қарастырылған.

Бұл жүйенің әзірленуі экологтар мен зерттеушілерге Каспий итбалықтарының популяциясын бақылауға қолдау көрсетуге бағытталған, бұл түрдің сақталуына және экологиялық деректерді басқару әдістерін жетілдіруге ықпал етеді.

АННОТАЦИЯ

Этот дипломный проект посвящён разработке мобильного приложения для мониторинга популяции каспийских тюленей с использованием технологий компьютерного зрения. Каспийские тюлени являются редким и уязвимым видом, численность которого сокращается под влиянием экологических факторов. Традиционные методы наблюдения обладают рядом ограничений, связанных со сложностью доступа и высокой трудоёмкостью процесса.

Предлагаемое решение включает использование дронов для съёмки территорий обитания тюленей и применения нейросетевой модели для автоматического распознавания и подсчёта особей на изображениях. Приложение базируется на клиент-серверной архитектуре, где анализ изображений выполняется на мобильном устройстве с помощью TensorFlow Lite, а данные сохраняются в локальной базе SQLite.

В работе рассматриваются современные методы распознавания объектов, включая bounding boxes и тепловые карты, а также существующие интерфейсы визуализации данных. Разработанная информационная система предоставляет пользователям удобный инструмент для сбора и анализа информации о популяции тюленей.

Приложение обладает простым и интуитивно понятным интерфейсом, возможностью загрузки изображений, визуализацией результатов анализа и экспортом данных. Предусмотрена оптимизация модели для работы в условиях ограниченных вычислительных ресурсов мобильных устройств.

Разработка данной системы направлена на поддержку экологов и исследователей в мониторинге численности каспийских тюленей, что способствует сохранению вида и улучшению методов управления экологическими данными.

ABSTRACT

This thesis project is dedicated to the development of a mobile application for monitoring the Caspian seal population using computer vision technologies. Caspian seals are a rare and vulnerable species whose population is declining due to environmental factors. Traditional observation methods have several limitations, including difficulties in access and the high labor intensity of the process.

The proposed solution involves using drones to capture images of seal habitats and applying a neural network model for automatic recognition and counting of individuals in the images. The application is based on a client-server architecture, where image analysis is performed on a mobile device using TensorFlow Lite, and the data is stored in a local SQLite database.

The study explores modern object recognition methods, including bounding boxes and heatmaps, as well as existing data visualization interfaces. The developed information system provides users with a convenient tool for collecting and analyzing data on the seal population.

The application features a simple and intuitive interface, the ability to upload images, visualization of analysis results, and data export. Model optimization is provided to ensure efficient performance under the limited computational resources of mobile devices.

The development of this system aims to support ecologists and researchers in monitoring the Caspian seal population, contributing to species conservation and improving environmental data management methods.

Реферат

Дипломный проект изложен на 49 страницах, содержит 39 рисунков, 2 таблицы и 2 приложения.

Ключевые слова: каспийский тюлень, мобильное приложение, мониторинг, компьютерное зрение, нейросеть, экология, распознавание объектов, защита животных.

Цель проекта — создание мобильного инструмента, позволяющего автоматизировать процесс сбора и анализа данных о популяции тюленей. В рамках решения используются дроны для фото- и видеосъёмки ареалов обитания, а также нейросетевые модели для автоматического распознавания и подсчёта особей на изображениях. Обработка данных осуществляется на мобильном устройстве с использованием TensorFlow Lite, что позволяет анализировать изображения без подключения к облачным сервисам. Результаты сохраняются в локальной базе данных SQLite.

Предложенное решение включает использование дронов для съёмки ареалов обитания тюленей и нейросетевой модели для автоматического распознавания и подсчёта особей. Обработка изображений осуществляется на мобильном устройстве с помощью TensorFlow Lite. Результаты сохраняются в локальной базе данных SQLite.

Приложение имеет интуитивно понятный интерфейс, поддерживает загрузку изображений, визуализацию анализа (bounding boxes, тепловые карты) и экспорт данных. Оптимизация модели обеспечивает стабильную работу в условиях ограниченных ресурсов мобильных устройств, что делает систему пригодной для полевых исследований.

Разработка направлена на поддержку экологов и специалистов в области охраны природы, предоставляя эффективный инструмент для сбора и анализа информации о популяции каспийских тюленей.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Мобильного приложения AI_TULEN, которое объединяет современные технологии искусственного интеллекта, компьютерного зрения и мобильной разработки для решения важной экологической задачи. Этот проект разработан с целью автоматизации процесса подсчёта и мониторинга популяции каспийских тюленей, которые относятся к уязвимым видам и нуждаются в защите.

С помощью дронов и нейросети приложение помогает идентифицировать тюленей на фотографиях и точно подсчитывать их количество. Оно обеспечивает удобный инструмент для учёных, экологов и специалистов в области охраны природы, заменяя традиционные ручные методы подсчёта, которые обычно требуют больших затрат времени и усилий.

В процессе реализации, проект прошёл через несколько ключевых этапов: от сбора и разметки данных до разработки, обучения и интеграции нейросетевой модели в мобильное приложение. Это приложение представляет собой уникальный пример того, как передовые технологии могут быть использованы для защиты экосистемы и улучшения методов научных исследований.

В своей работе использовались предобученные архитектуры нейросетей, такие как EfficientDet и SSD MobileNet, адаптированные для задачи детекции объектов. Обучение модели проводилось на платформе Google Colab с использованием размеченных данных, созданных вручную. Завершённая модель была конвертирована в формат TensorFlow Lite для внедрения в Android-приложение.

Результат работы — приложение, которое, помимо своей функциональности, демонстрирует возможности современных технологий в решении социальных и экологических задач.

1 Анализ предметной области и постановка задачи

1.1 Обзор проблемы подсчёта каспийских тюленей

Каспийские тюлени — уникальный и уязвимый вид, который обитает исключительно в Каспийском море. В последние годы их численность значительно сократилась из-за ухудшения экологических условий, браконьерства и изменения климата. Мониторинг популяции тюленей является важной задачей для сохранения вида и обеспечения экологического равновесия.

Традиционные методы подсчёта тюленей включают визуальное наблюдение с использованием кораблей или наземных станций. Эти методы обладают рядом ограничений:

Сложность доступа: Тюлени обитают в труднодоступных районах, часто на льдинах, что делает наблюдение затруднительным.

Низкая эффективность: Ручной подсчёт объектов на больших территориях требует значительных временных и трудовых затрат.

Погрешности: Погодные условия, такие как туман или плохая освещённость, могут снижать точность наблюдений.

Использование дронов для съёмки больших территорий с высоты решает проблему доступа, однако требует автоматизации процесса анализа данных. В данном проекте предлагается мобильное приложение, оснащённое нейросетью, способной распознавать каспийских тюленей на фотографиях и подсчитывать их количество. Это решение обещает значительно повысить эффективность и точность мониторинга.

1.2 Существующие подходов к визуализации к визуализации каспийских тюленей

На сегодняшний день существуют некоторые методы визуализации и анализа данных, которые могут быть адаптированы для мониторинга популяции животных, включая (рисунок 1):

Интерактивные карты:

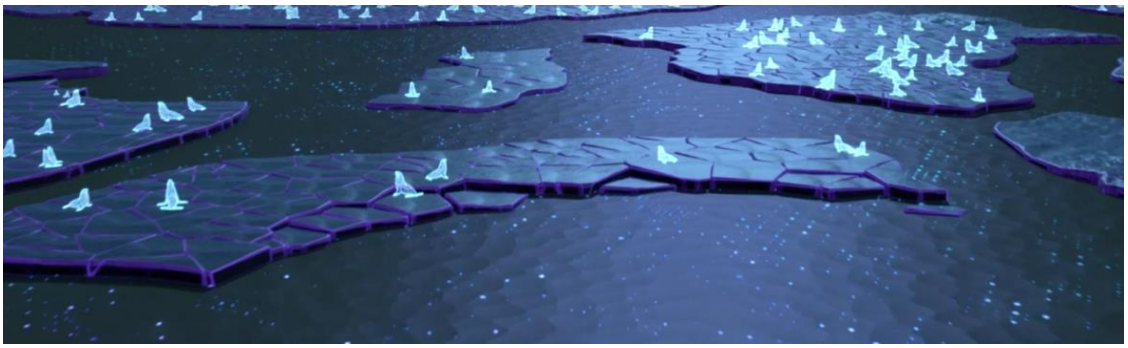


Рисунок 1 – Интерактивные карты

Используются для отображения территориального распределения объектов, таких как места скопления тюленей.

Позволяют отслеживать изменения популяции в зависимости от сезона или других факторов.

Рамки на изображениях:

Методы выделения bounding boxes, которые автоматически определяют объекты на фото.

Эти рамки помогают идентифицировать каждого тюленя и его положение.

Мобильные приложения для распознавания объектов:

Например, приложения для мониторинга дикой природы на основе компьютерного зрения.

Эти подходы демонстрируют успех в визуализации данных, однако их применение для распознавания тюленей требует адаптации и создания специализированных решений.

1.2.1 Анализ методов распознавания животных

Методы распознавания животных, включая каспийских тюленей, основаны на технологии компьютерного зрения. Среди наиболее эффективных:

Обученные нейросети для детекции объектов:

Архитектуры EfficientDet и SSD MobileNet V2 демонстрируют высокую точность при распознавании объектов на сложных изображениях.

Тепловые карты:

Эти карты могут отображать плотность распределения животных по территории.

Машинное обучение на размеченных данных:

Использование bounding boxes и классов объектов для обучения нейросетей на больших наборах данных.

Применение таких методов позволяет быстро анализировать изображения, делать точные прогнозы и упрощать мониторинг.

1.2.2 Примеры интерфейсов для отображения данных

В рамках анализа существующих решений можно выделить:

Приложения и интерфейсы, предоставляющие данные о популяции животных в реальном времени (рисунок 2). Например, системы мониторинга диких животных с использованием GPS и камер.

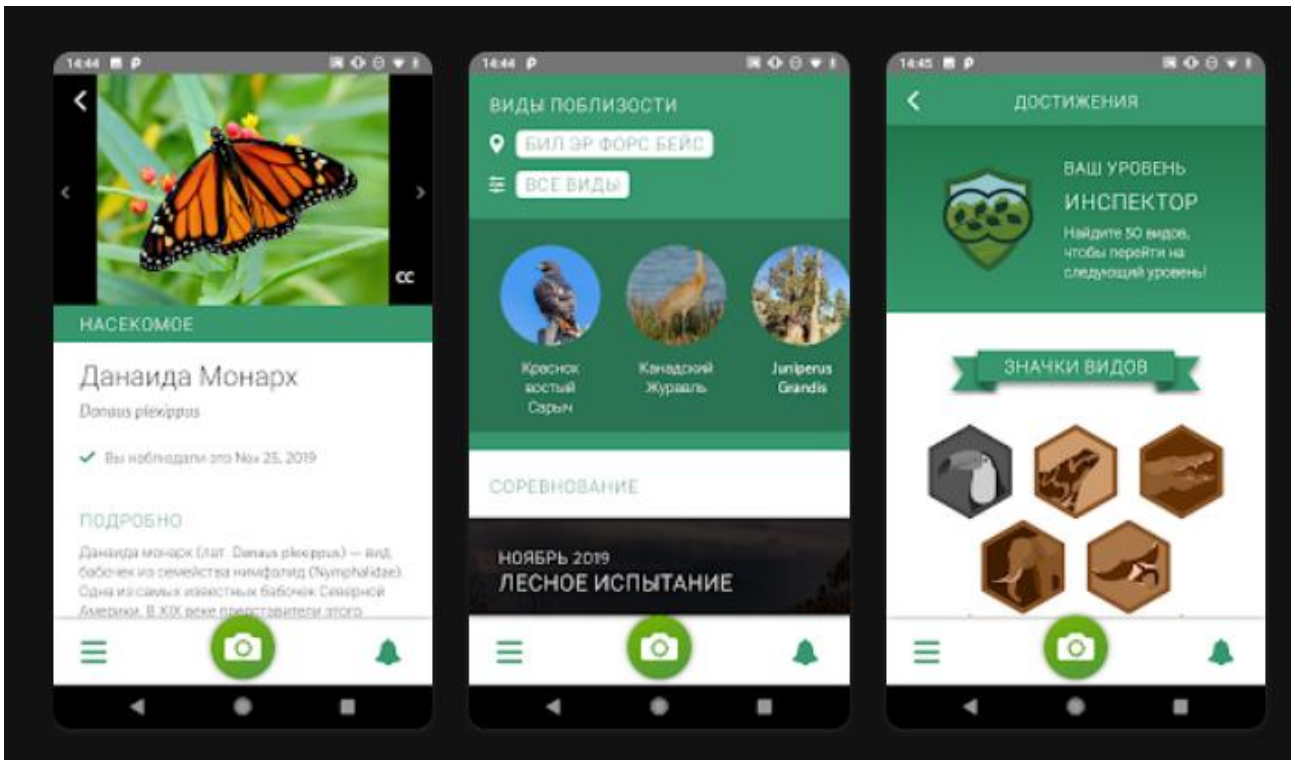


Рисунок 2 – Аналог мобильного приложения мониторинга животных

Простые интерфейсы мобильных приложений:

Например, приложения для аграрного мониторинга, где пользователь получает данные о животных через камеру или фотографии из галереи. Такие приложения обладают лёгкой навигацией и предоставляют визуальную информацию.

1.2.3 Оценка сильных и слабых сторон существующих решений

Сильные стороны приложения AI_Tulen:

Автоматизация анализа данных:

Нейросети позволяют заменить ручной анализ изображений.

Интеграция с мобильными устройствами:

Приложения удобны для использования в полевых условиях.

Гибкость технологии:

Возможность адаптации под разные виды животных.

Слабые стороны приложения AI_Tulen:

Ограничение по обучению:

Нейросети требуют размеченных данных высокого качества.

Точность в сложных условиях:

Сложные условия съёмки, такие как плохое освещение (рисунки 3 и 4), могут снижать эффективность модели.

На этих фотографиях можно видеть как и съемку с обычной камеры, так и с дронов.

Рисунок 3 – Фотография стада тюленей снятая с камеры

Рисунок 4 – Фотография стада тюленей снятая с дрона

Эти оценки демонстрируют необходимость адаптации и расширения существующих решений для задач мониторинга каспийских тюленей.

1.2.4 Обзор существующих технологий, их оценка и выбор подходящего решения

В рамках разработки мобильного приложения AI_TULEN были применены современные технологии, обеспечивающие удобство

использования, высокую производительность и точность анализа изображений. В данном разделе рассмотрены основные технологии, их плюсы и минусы, а также причины выбора конкретных решений.

1.2.4.1 Технологии, использованные в проекте

Язык программирования: Java (Android)
Плюсы:
- Поддержка Android
- Большое количество библиотек
- Высокая производительность

Минусы:
- Длительное время разработки
- Требует большего количества кода по сравнению с Kotlin

Почему выбран Java?

Java является основным языком для разработки Android-приложений, обеспечивая совместимость с существующими библиотеками и стабильную работу на различных устройствах.

База данных: SQLite
Плюсы:
- Локальное хранение данных
- Низкое потребление ресурсов
- Простота использования

Минусы:
- Ограниченная масштабируемость
- Нет встроенной поддержки облачного хранения

Почему выбрана SQLite?

SQLite позволяет эффективно сохранять результаты анализа изображений, обеспечивая быстрый доступ к истории данных в приложении.

Обработка изображений: OpenCV
Плюсы:
- Высокая точность анализа изображений
- Поддержка машинного обучения
- Гибкость в обработке графических данных

Минусы:
- Требует дополнительной оптимизации для мобильных устройств
- Сложность интеграции с некоторыми API

Почему выбран OpenCV?

OpenCV позволяет анализировать изображения, выявлять объекты (например, тюленей) и получать количественные данные о них.

Взаимодействие с сервером: OkHttp

Плюсы:

- Простота работы с HTTP-запросами
- Высокая производительность
- Легкость интеграции

Минусы:

- Требует ручной обработки ошибок
- Возможны задержки при медленном интернет-соединении

Почему выбран OkHttp? OkHttp позволяет быстро отправлять изображения на сервер и получать результаты анализа.

Облачные технологии: Google Colab

Плюсы:

- Высокая масштабируемость
- Быстрая обработка данных
- Удобная интеграция

Минусы:

- Зависимость от интернет-соединения
- Потребность в дополнительных настройках безопасности

Почему выбран Google Colab?

Google Colab обеспечивает надежное хранение и обработку данных, позволяя выполнять анализ изображений на мощных серверах.

1.2.4.2 Выбор технологий

На основе оценки плюсов и минусов использованных технологий было принято решение о применении **Java, SQLite, OpenCV, OkHttp и Google Colab**, так как они обеспечивают:

- **Стабильность и производительность**
- **Простоту интеграции**
- **Точность анализа изображений**
- **Гибкость в развитии проекта**

1.2.4.3 Применение Make Sense и YOLO в AI_TULEN

В нашем проекте Make Sense использовался для разметки изображений с тюленями, а затем данные экспортировались в YOLO для обучения модели. Это обеспечило:

Создание качественного датасета для обучения нейросети.

Выделение контуров каждого тюленя на изображении.

Определение количества объектов на фото.

Процесс работы:

1. Загружаем изображение в Make Sense.
2. Выделяем контуры каждого тюленя.
3. Присваиваем объектам категорию ("seal").

4. Экспортируем данные в формат YOLO и передаем их модели для обучения.

Оценка Make Sense и YOLO: плюсы и минусы

Make Sense

Плюсы:

- Интуитивно понятный интерфейс
- Поддержка нескольких форматов экспорта
- Автоматическое выделение объектов

Минусы:

- Требуется ручная работа при сложных изображениях
- Возможны ошибки при неправильной сегментации
- Не подходит для обработки больших объемов изображений без автоматизации

Почему выбран Make Sense?

Make Sense был выбран, потому что он позволяет точно разметить изображения для машинного обучения, интегрируется с OpenCV и TensorFlow и обеспечивает качественные данные для модели AI_TULEN.

YOLO (You Only Look Once)

Плюсы:

- Высокая скорость работы при детекции объектов
- Компактный формат хранения аннотаций
- Совместимость с популярными моделями машинного обучения

Минусы:

- Требует точной нормализации координат
- Не поддерживает сложные многоугольные сегменты объектов

Почему выбран YOLO?

YOLO был выбран, потому что он оптимален для детекции объектов (тюленей), позволяет обучать нейросети на реальных данных, размеченных в Make Sense, и совместим с TensorFlow, PyTorch и OpenCV.

1.3 Постановка задачи

Для успешного мониторинга численности каспийских тюленей необходимо разработать приложение, которое:

Распознаёт тюленей на фотографиях, снятых с дронов, с высокой точностью.

Автоматически подсчитывает количество тюленей на изображении.

Обеспечивает удобный интерфейс для пользователей.

1.3.1 Техническое задание для ПО

Разработать систему компьютерного зрения для распознавания объектов (тюленей) на изображениях.

Обеспечить высокую точность подсчёта объектов при обработке снимков.

1.3.2 Функциональные требования к дизайну

Простой и интуитивно понятный интерфейс для загрузки фотографий с дронов.

Визуализация результатов в виде рамок и численного подсчёта объектов.

1.3.3 Нефункциональные требования

Оптимизация модели для работы на мобильных устройствах с ограниченными ресурсами.

Поддержка обработки изображений в реальном времени.

2 Проектирование информационной системы

В данном разделе представлены ключевые аспекты проектирования информационной системы, включая её архитектуру, структуру базы данных, дизайн интерфейса и обоснование выбора технологий.

2.1 Проектирование архитектуры ИС

Архитектура приложения базируется на клиент-серверной модели, где обработка изображений и нейросетевой анализ происходят непосредственно на мобильном устройстве. Это обеспечивается интеграцией модели TensorFlow Lite.

Описание архитектуры:

Мобильный клиент:

Пользователь загружает изображение, снятое дроном.

Локально запускается инференс модели TensorFlow Lite для распознавания объектов (тюленей).

Серверная часть (по необходимости):

Используется для хранения данных и сбора статистики по обработанным изображениям.

Обеспечивает обновление моделей, если это потребуется в будущем.

Модули системы:

Интерфейс пользователя (UI): Предоставляет удобную платформу для загрузки изображений, просмотра результатов и сохранения данных.

Модуль нейросети: Реализован с использованием TensorFlow Lite для Java.

База данных: Хранит информацию о результатах распознавания.

Архитектурная схема (рисунок 5):

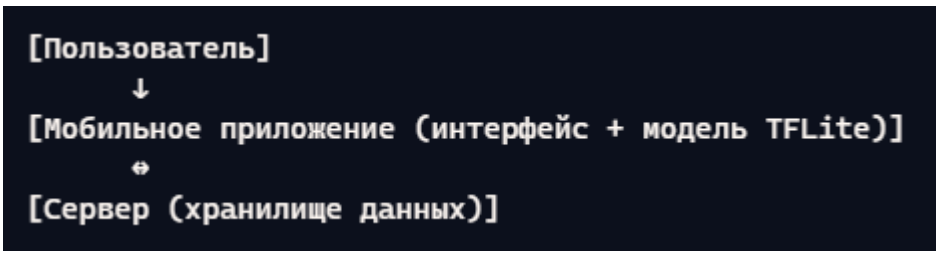


Рисунок 5 – Архитектурная схема

2.2 Структура БД

Для хранения результатов анализа и статистики используется структура базы данных (рисунок 6), которая включает следующие таблицы:

Таблица: Изображения

Поле	Тип	Описание
`id`	INTEGER	Уникальный идентификатор
`image_name`	VARCHAR	Имя файла изображения
`upload_date`	TIMESTAMP	Дата и время загрузки
`tuleni_count`	INTEGER	Количество распознанных тюленей

Таблица: Объекты

Поле	Тип	Описание
`id`	INTEGER	Уникальный идентификатор
`image_id`	INTEGER	ID изображения
`bbox`	VARCHAR	Bounding box (координаты рамки)
`confidence`	FLOAT	Уверенность модели (%)

Рисунок 6 – Структура базы данных

Пример кода для создания базы данных:

```
CREATE TABLE Images (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    image_name VARCHAR (255) NOT NULL,
    upload_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    tuleni_count INTEGER
);

CREATE TABLE Objects (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    image_id INTEGER,
    bbox VARCHAR (255),
    confidence FLOAT,
    FOREIGN KEY (image_id) REFERENCES Images(id)
);
```

2.3 Разработка дизайна ИС

Дизайн интерфейса приложения ориентирован на простоту и удобство пользователя (рисунок 7).

Главный экран:

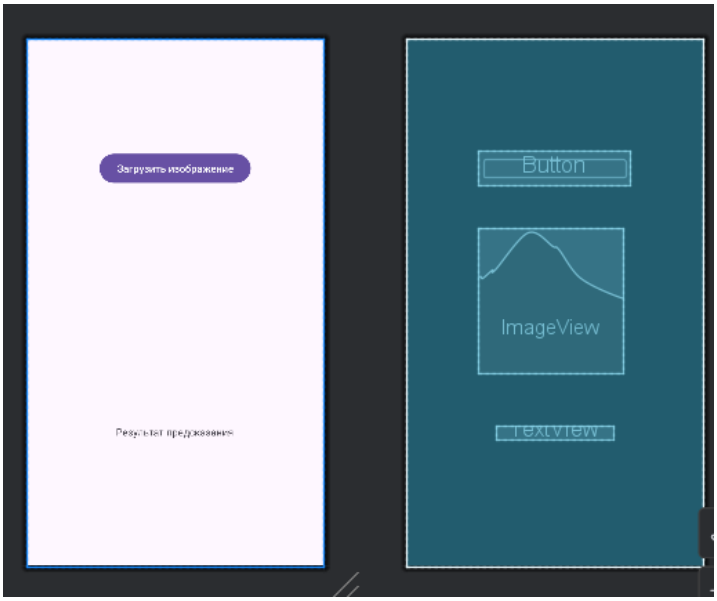


Рисунок 7 – пример главный экран

Возможность загрузки изображения с телефона или камеры дрона (рисунок 8).

Кнопка для запуска обработки.

Экран результатов:

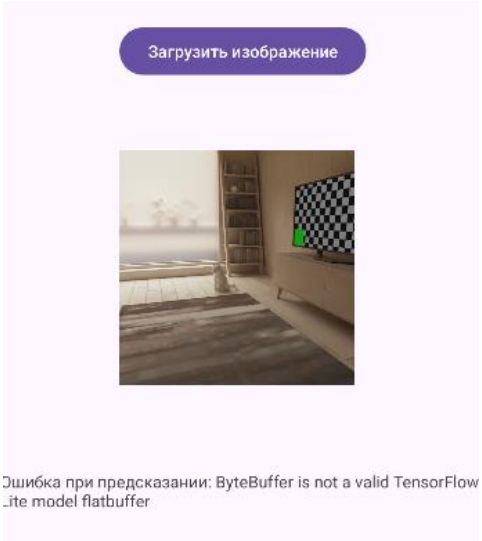


Рисунок 8 – Пример экрана результата

Отображение изображения с выделенными рамками вокруг тюленей.
Число обнаруженных объектов (тюленей).

2.4 Обоснование выбора технологий и инструментов

Для реализации приложения были выбраны следующие технологии:

TensorFlow Lite:
Лёгкость интеграции с мобильными платформами (Android)(рисунок 9).
Высокая скорость инференса на устройствах с ограниченными вычислительными ресурсами.

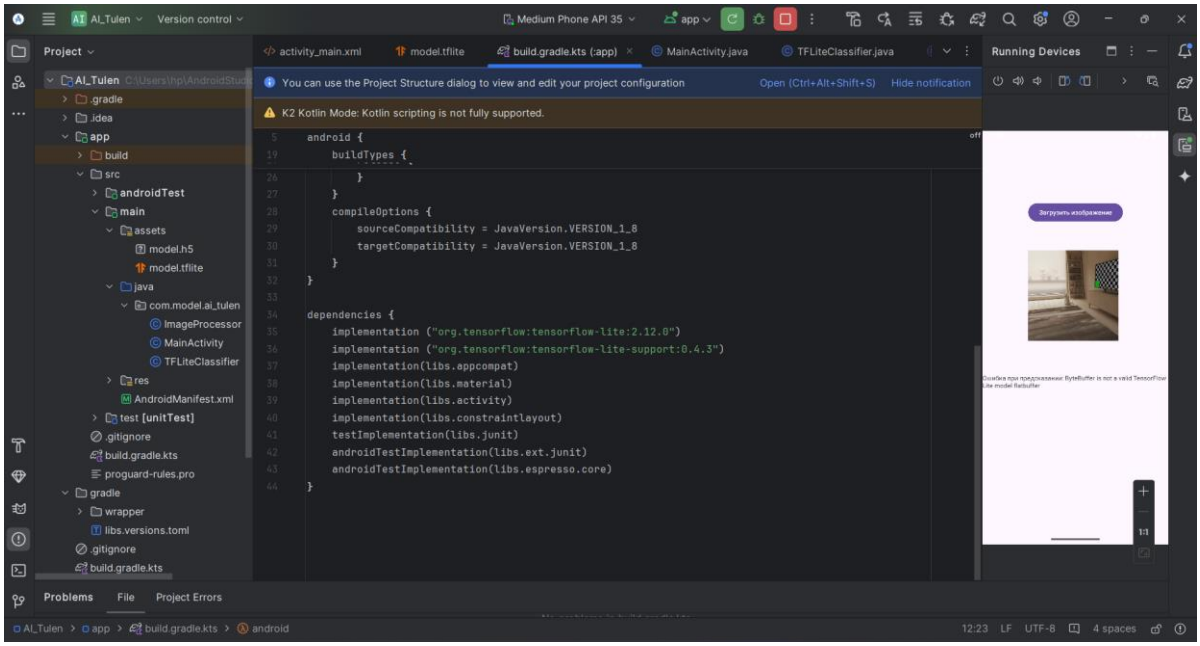


Рисунок 9 – пример подключения TendorFlow Lite

Android Studio:
Платформа для разработки Android-приложений с поддержкой Java и Kotlin.
Google Colab:
Используется для обучения и тестирования нейросетевой модели (рисунок 10). Обеспечивает мощные вычислительные ресурсы для обработки данных.
SQLite:
Встроенная база данных для хранения результатов анализа на мобильном устройстве.
Пример кода: Интеграция TensorFlow Lite в Android
Шаг 1: Добавьте модель в проект
Переместите файл seal_model.tflite в папку /assets вашего проекта в Android Studio.

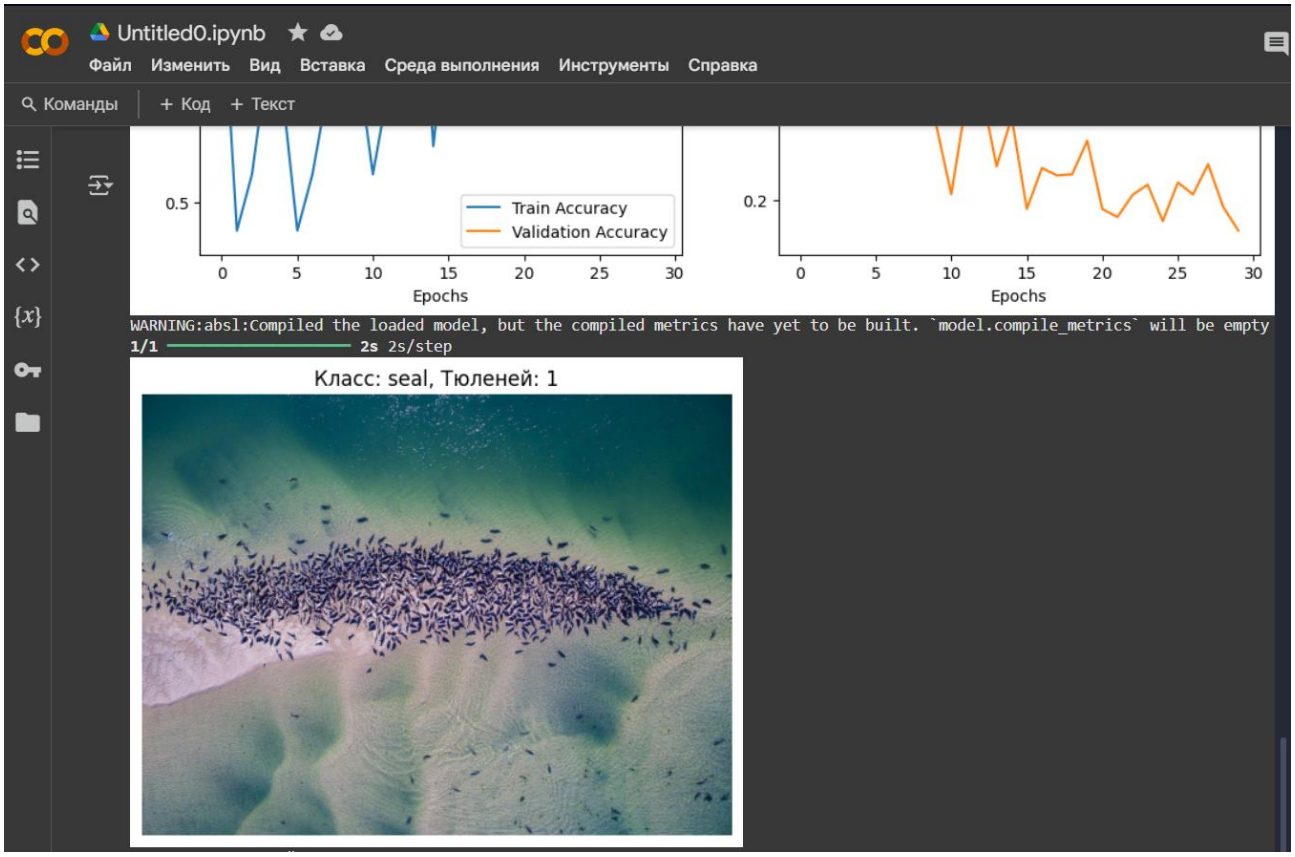


Рисунок 10 – Обучение модели

Шаг 2: Подключите библиотеку TensorFlow Lite
В файле build.gradle добавьте:
gradle
implementation 'org.tensorflow:tensorflow-lite:2.10.0'
implementation 'org.tensorflow:tensorflow-lite-support:0.4.3'
Шаг 3: Реализуйте инференс модели
java
import org.tensorflow.lite.Interpreter;.

.

3 Разработка ИС

В процессе разработки **AI_TULEN** были выполнены ключевые этапы, позволяющие создать функциональное и удобное приложение для анализа изображений тюленей.

3.1 Процесс разработки

Основные этапы разработки:
Формулирование требований:

Определили функционал приложения (таблица 1), включая обработку изображений, разметку данных и взаимодействие с сервером.

Таблица 1 – Функционал приложений

№	Функция	Описание	Используемые технологии
1	Загрузка изображений	Позволяет пользователю загружать изображения для анализа.	Android SDK, Java
2	Анализ изображений	Распознает тюленей и определяет их количество.	OpenCV, YOLO, TensorFlow
3	Разметка данных	Выделение контуров объектов (тюленей) на изображении.	Make Sense, JSON, YOLO Format
4	Сохранение результатов анализа	Хранение данных о распознанных объектах и их количестве.	SQLite
5	Обмен данными с сервером	Отправка изображений на сервер и получение результатов.	OkHttp, REST API
6	Облачное хранилище данных	Обеспечивает сохранение изображений и результатов анализа в облаке.	Google Colab, Firebase
7	Просмотр истории анализов	Позволяет пользователю видеть ранее загруженные изображения и результаты их анализа.	SQLite, RecyclerView
8	Очистка базы данных	Удаление всех сохраненных анализов по запросу пользователя.	SQLite, Room Database
9	Пользовательский интерфейс	Интуитивно понятный UI для взаимодействия с приложением.	XML Layout, Material Design

Выбор технологий:

Провели анализ существующих решений и выбрали Java, SQLite, OpenCV, OkHttp, Google Colab и Make Sense (рисунок 11).

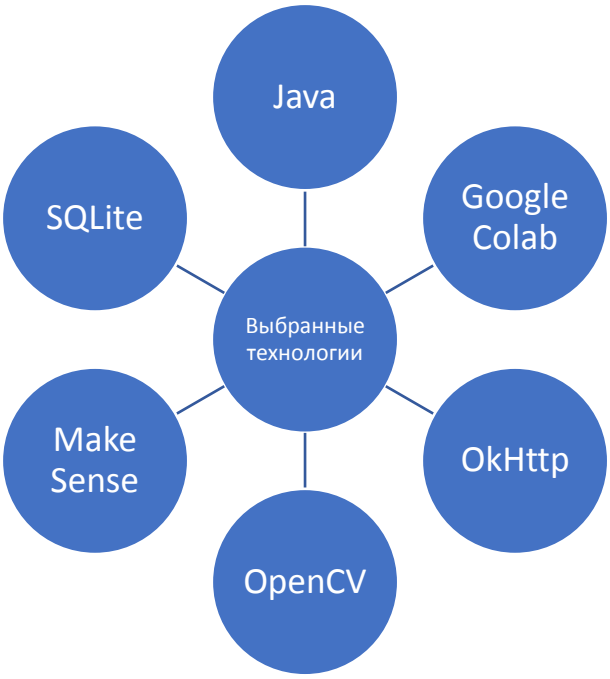


Рисунок 11 – Выбор решений

Разработка архитектуры:

Определили, как будут взаимодействовать компоненты приложения (таблица 2) — пользовательский интерфейс, база данных и серверная обработка изображений.

Таблица 2 – Компоненты решений

№	Компонент	Описание	Используемые технологии
1	Пользовательский интерфейс (UI)	Графический интерфейс для взаимодействия с пользователем.	XML Layout, Material Design, Android View
2	Логика приложения	Обработка пользовательских действий, управление UI, обработка данных.	Java, MVVM
3	База данных	Хранение данных о результатах анализа изображений.	SQLite, Room Database
4	Обработка изображений	Анализ загруженных изображений, выделение объектов (тюленей).	OpenCV, YOLO, TensorFlow

№	Компонент	Описание	Используемые технологии
5	Разметка данных	Разметка изображений для обучения моделей машинного обучения.	Make Sense, YOLO Format
6	Обмен данными с сервером	Отправка изображений на сервер, получение результатов анализа.	OkHttp, REST API
7	Облачное хранилище	Сохранение изображений и результатов в облаке.	Google Colab, Firebase
8	Тестирование и отладка	Проверка корректности работы приложения и его компонентов.	JUnit, Logcat, Android Debugging

Создание пользовательского интерфейса:

Разработали дизайн и основы экрана приложения в Android Studio (рисунок 12).

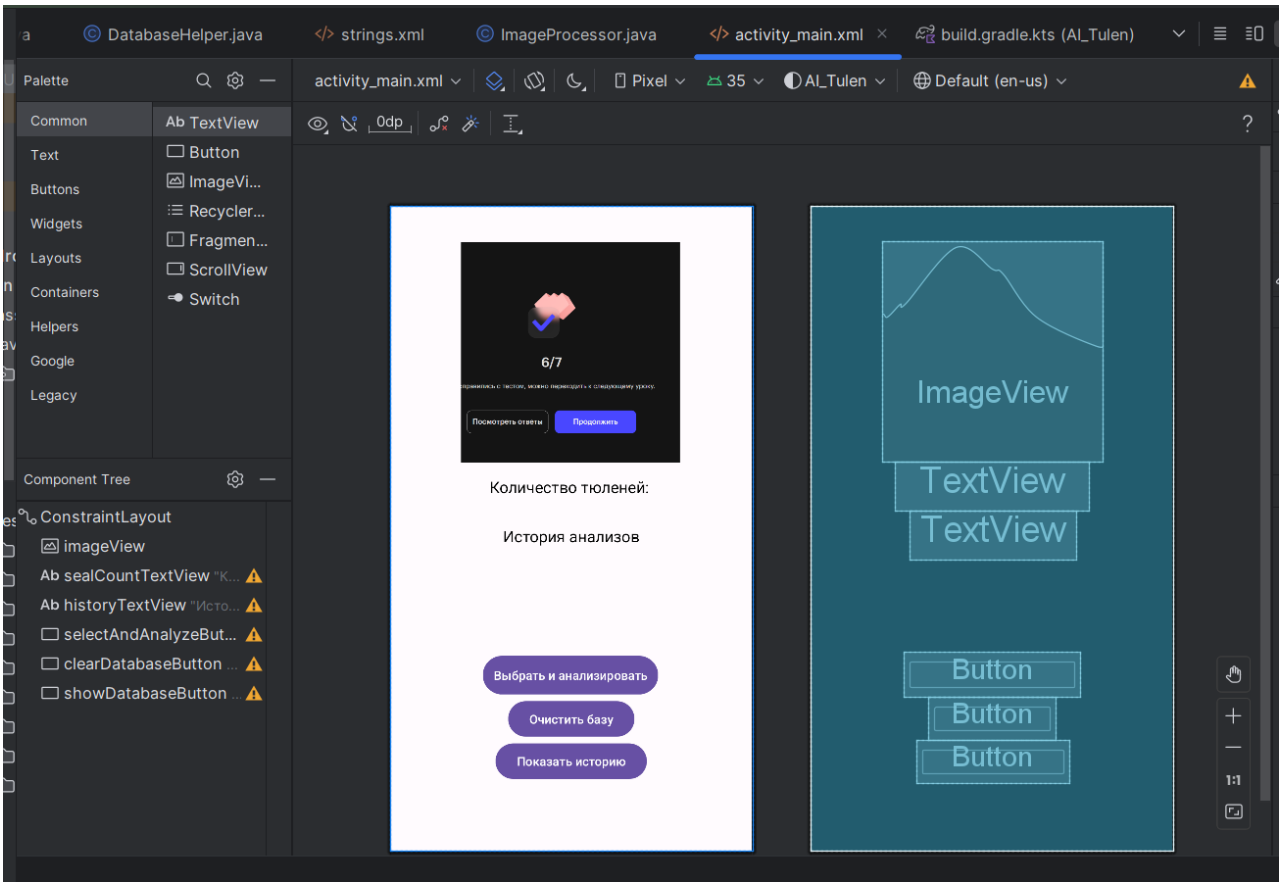


Рисунок 12 – Дизайн экрана приложения

Интеграция с сервером:

Настроили обмен данными между приложением и сервером с использованием OkHttp (рисунок 13).

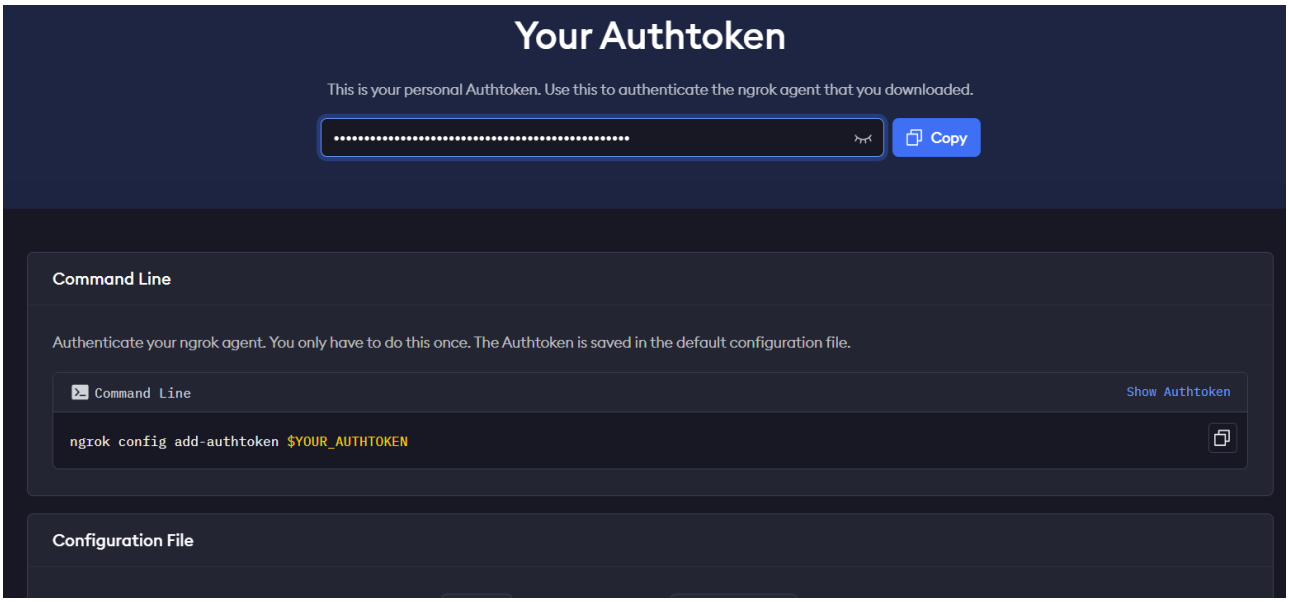


Рисунок 13 – Обмен данными

```
#    Установка необходимых библиотек
!pip install flask pyngrok ultralytics
# □ Импорт библиотек
from flask import Flask, request, jsonify
from pyngrok import ngrok
import os
import uuid
import logging
from google.colab import drive
from ultralytics import YOLO
import gc  # Автоочистка памяти
# □ Подключение Google Диска
drive.mount('/content/drive', force_remount=True)
# □ Настройка Ngrok для онлайн-доступа
NGROK_AUTH_TOKEN = "********"
ngrok.set_auth_token(NGROK_AUTH_TOKEN)
# □ Проверяем наличие модели перед загрузкой
model_path = "/content/drive/MyDrive/seal_detector.pt"
if not os.path.exists(model_path):
    raise FileNotFoundError(f"□ Модель не найдена по пути: {model_path}. Запустите `train_model.py`!")
# □ Загружаем **обученную** модель
model = YOLO(model_path)
# □ Проверяем `data.yaml`, загружены ли классы
with open("/content/drive/MyDrive/dataset/data.yaml", "r") as f:
    data_yaml = f.read()
```

```
if "seal" not in data_yaml:
    raise ValueError("□ Ошибка: `seal` не найден в data.yaml! Проверьте разметку.")
# □ Получаем классы модели и проверяем, загружены ли они
class_names = model.names if hasattr(model, 'names') else model.model.names
if not class_names:
    raise ValueError("□ Ошибка: YOLO не загружает классы! Проверьте data.yaml.")
print("□ Загруженные классы:", class_names)
# □ Папка для временного хранения изображений
IMAGE_FOLDER = "/content/drive/MyDrive/dataset/train/images"
os.makedirs(IMAGE_FOLDER, exist_ok=True)  # Создаем папку, если её нет
# □ Flask API
app = Flask(__name__)
@app.route('/')
def home():
    return '''
    <!doctype html>
    <html>
    <head><title>Seal Detection</title></head>
    <body>
      <h1>Выберите изображение для анализа</h1>
      <form method="POST" action="/analyze" enctype="multipart/form-data">
        <input type="file" name="image">
        <input type="submit" value="Загрузить и анализировать">
      </form>
    </body>
    </html>
    '''
@app.route('/analyze', methods=['POST'])
def analyze():
    try:
      file = request.files.get('image')
      if not file:
        return jsonify({"error": "Файл не загружен"}), 400
      # □ Сохранение загруженного изображения
      unique_filename = f"temp_{uuid.uuid4()}.jpg"
      file_path = os.path.join(IMAGE_FOLDER, unique_filename)
      file.save(file_path)
      # □ Анализ изображения с YOLO (правильные параметры!)
      results = model.predict(source=file_path, conf=0.3, imgsz=640)

      # □ Подсчет тюленей (избегаем ошибки `int(obj[-1])`)
      seal_count = sum(1 for obj in results[0].boxes.data if obj[2] > 50 and
class_names[int(obj[-1])] == "seal")
      #    Очистка памяти и удаление временного файла
      del results
      gc.collect()
      os.remove(file_path)
      return jsonify({"seal_count": seal_count})
    except Exception as e:
      return jsonify({"error": str(e)}), 500
```

```
# 🚀 Запуск сервера
if __name__ == "__main__":
    port = 5000
    public_url = ngrok.connect(port)
    print(f"🌍 Public URL: {public_url}")
    app.run(host="0.0.0.0", port=port)
```

Обучение модели:

При обучении были применен следующие фотографии (рисунок 13).

Провели разметку изображений в Make Sense (рисунки 14-19) и экспортировали данные в YOLO для машинного обучения (рисунки 20-21).

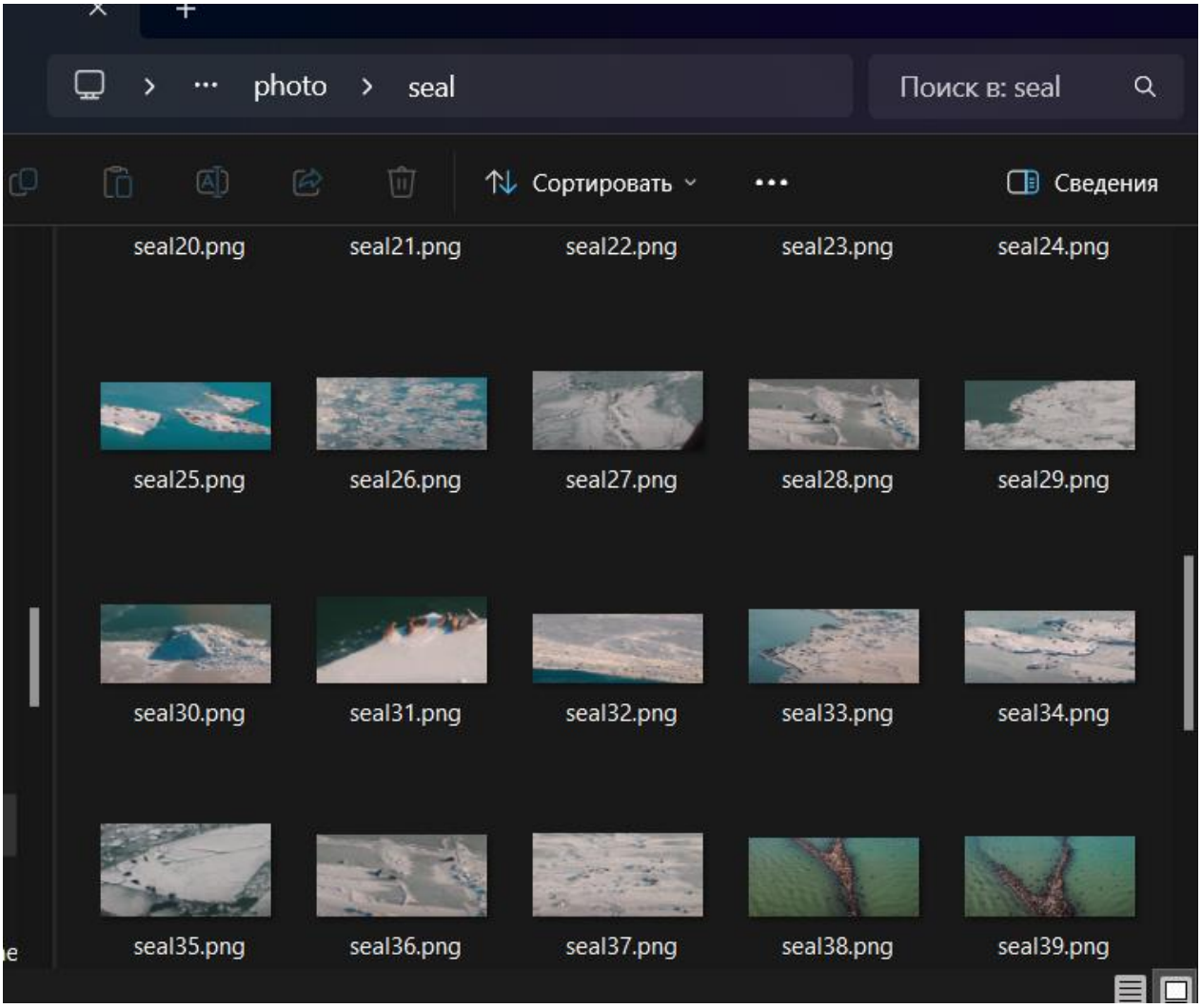


Рисунок 13 – Фотографии используемые при обучении

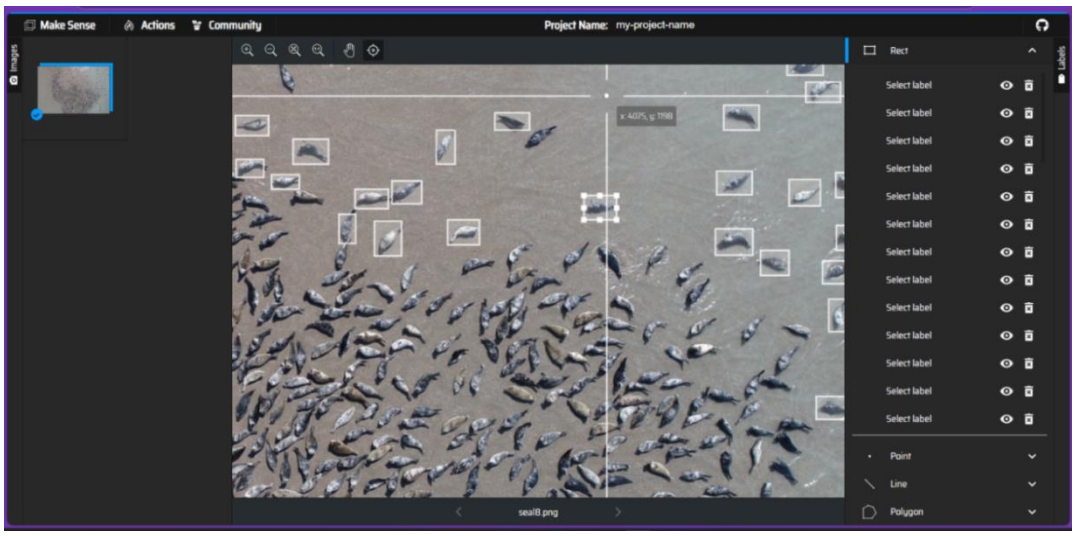


Рисунок 14 – Разметки фотографий для обучения

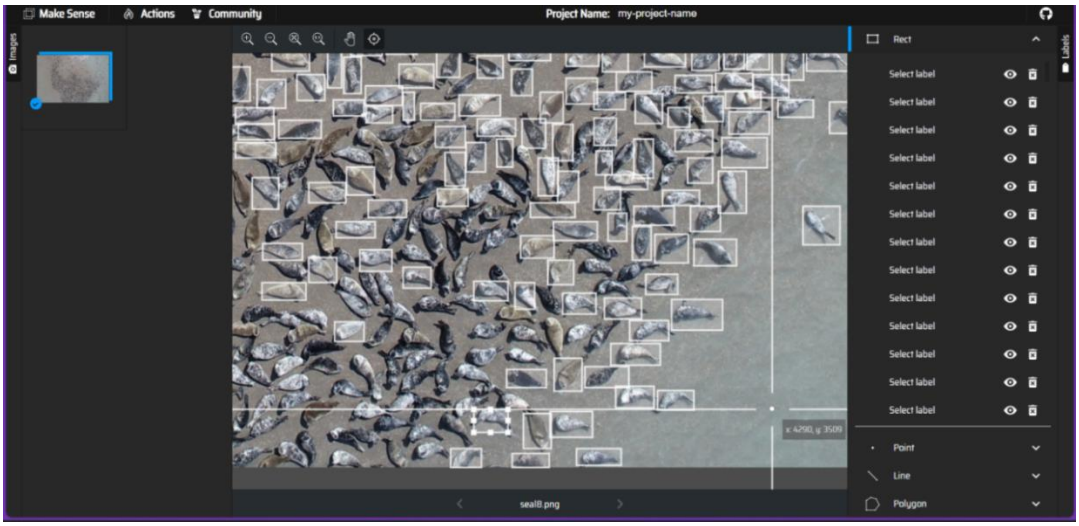


Рисунок 15 – Разметки фотографий для обучения

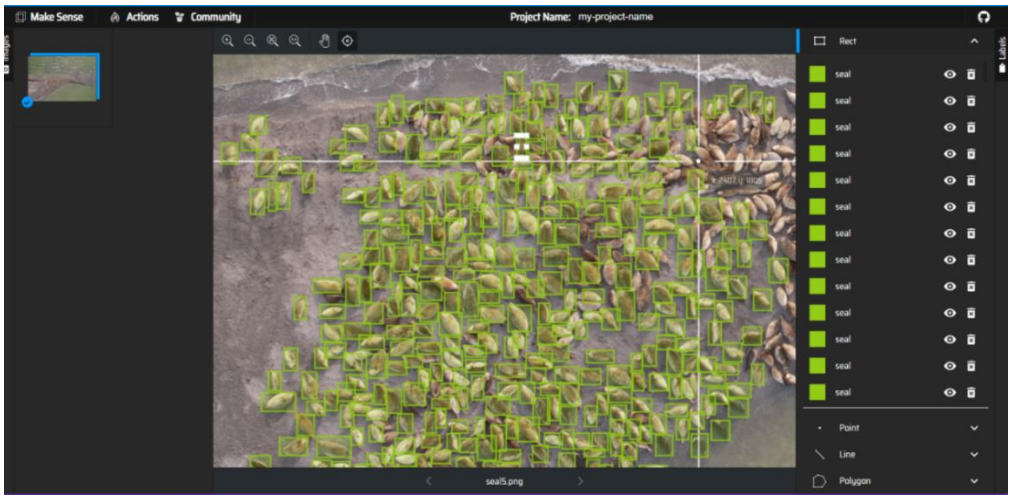


Рисунок 16 – Разметки фотографий для обучения

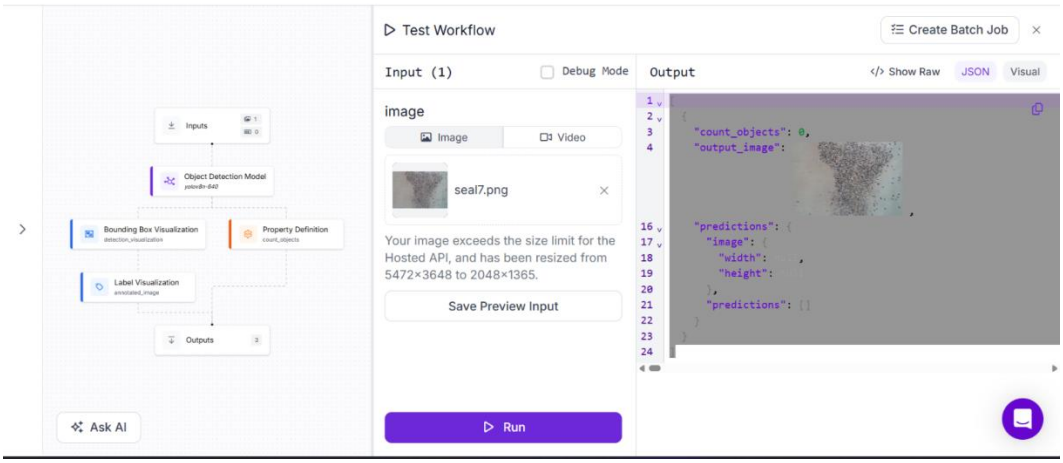


Рисунок 17 – Разметки фотографий для обучения

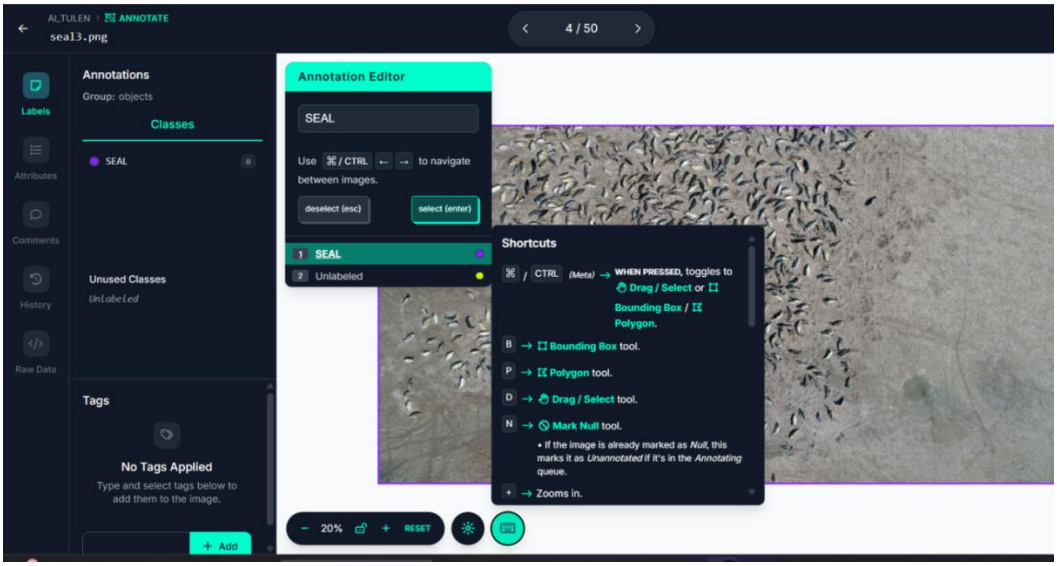


Рисунок 18 – Разметки фотографий для обучения

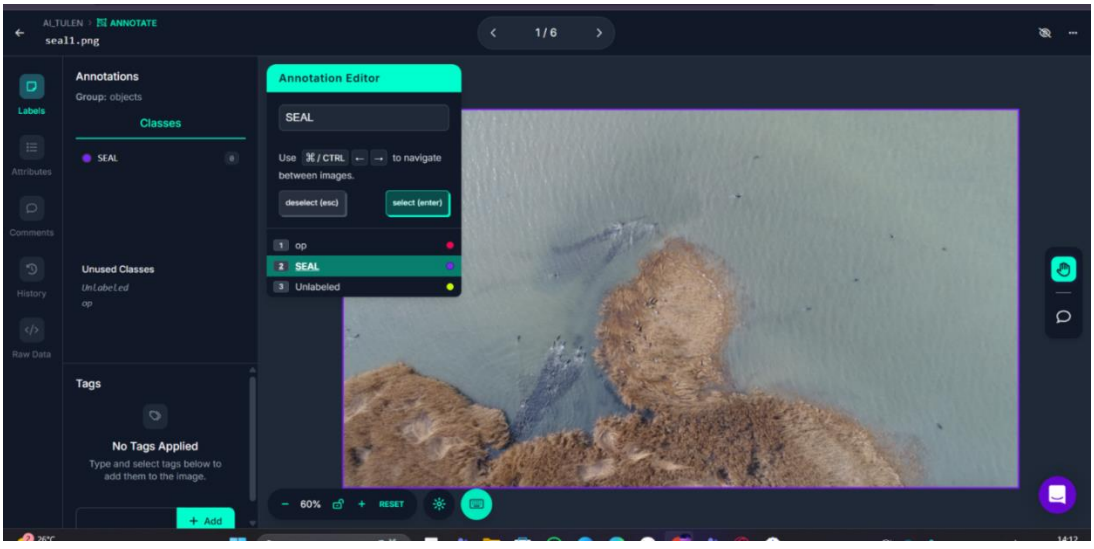


Рисунок 19 – Разметки фотографий для обучения

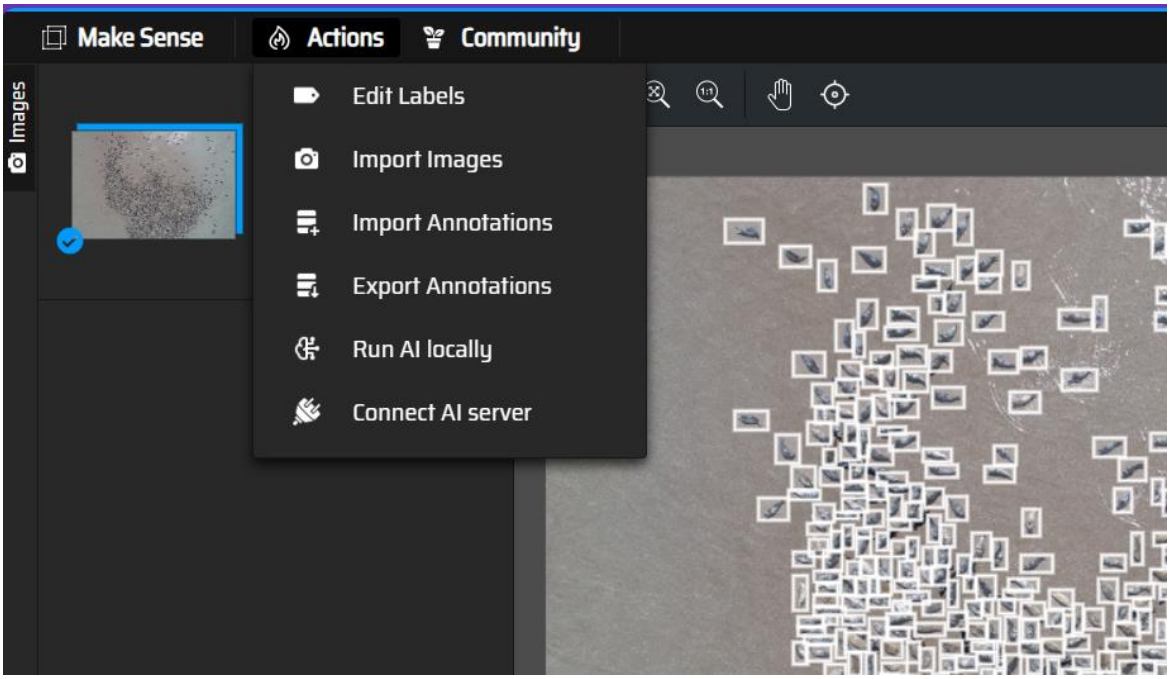


Рисунок 20- Экспорт размеченных фотографий

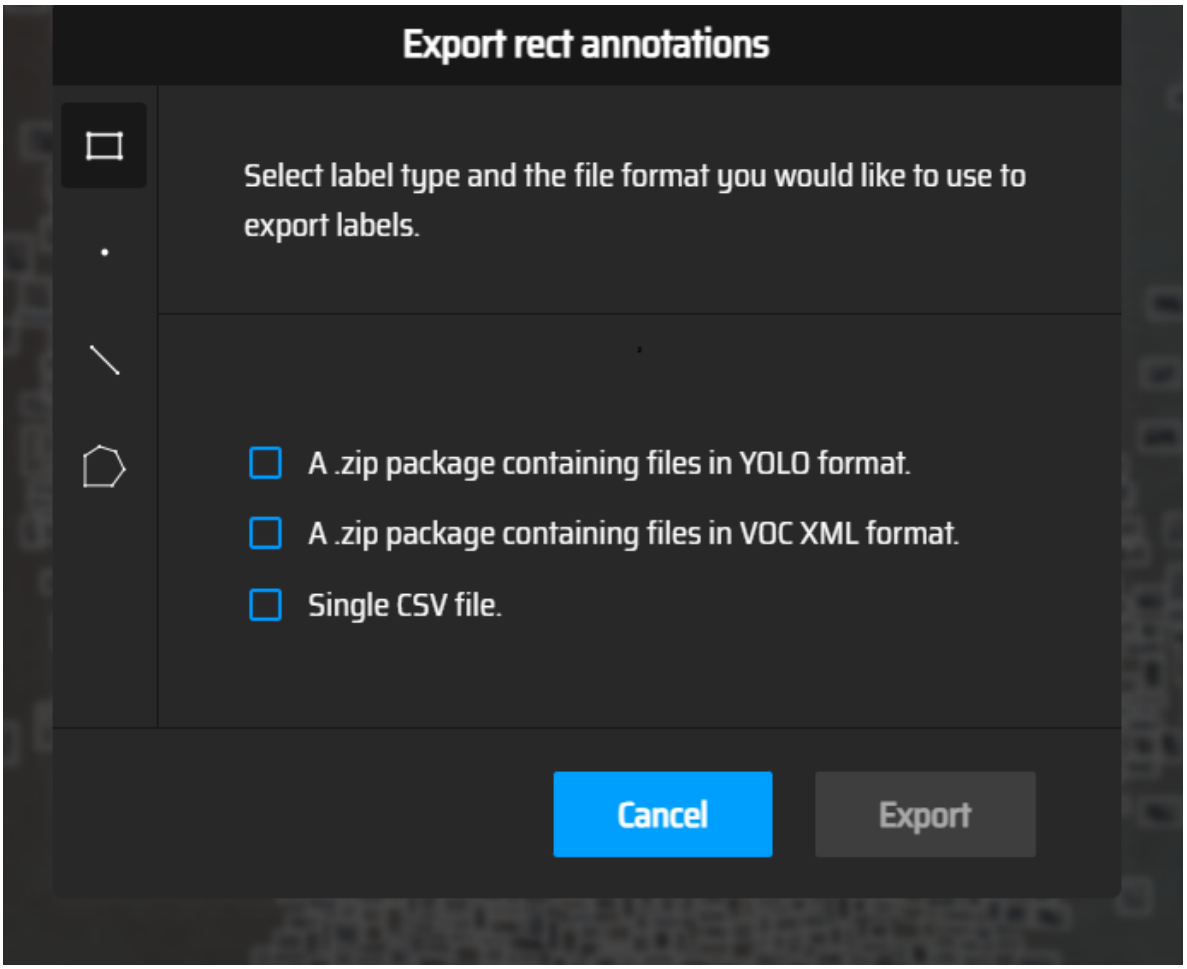


Рисунок 21- Экспорт размеченных фотографий

Оптимизация:

Улучшили работу приложения для стабильного функционирования на различных устройствах.

Процесс выбора и анализ фотографий с сайта представлен на рисунках 22-25.

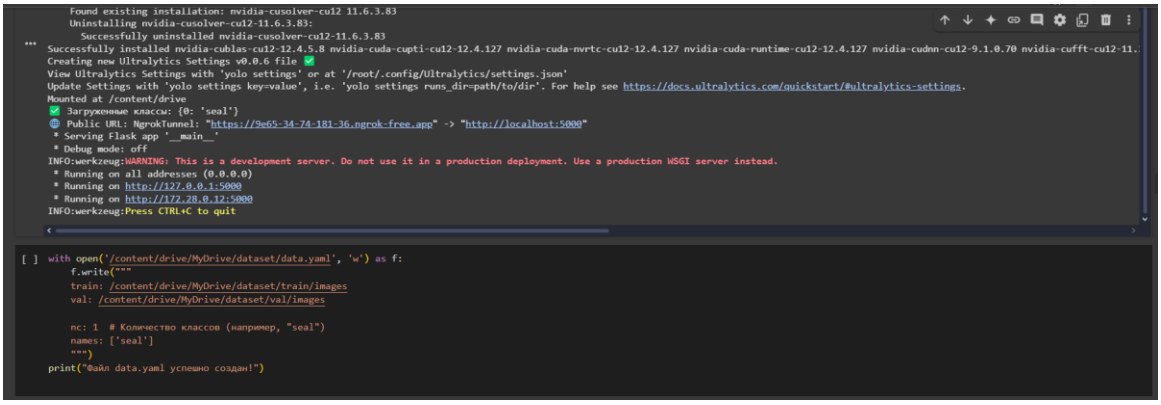


Рисунок 22 – Домен сайта обученной модели

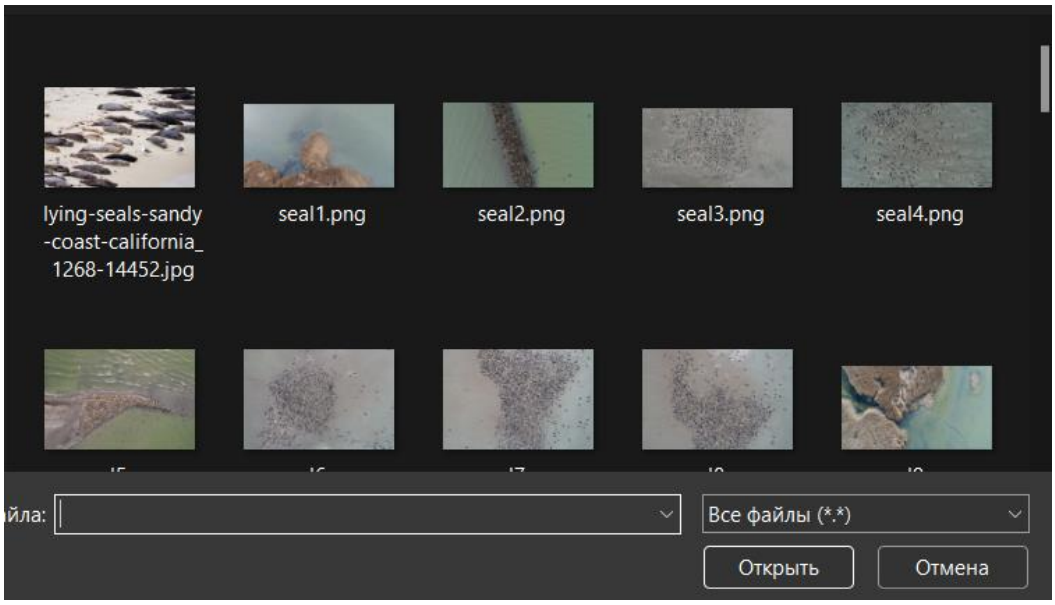


Рисунок 23 – Выбор фотографий

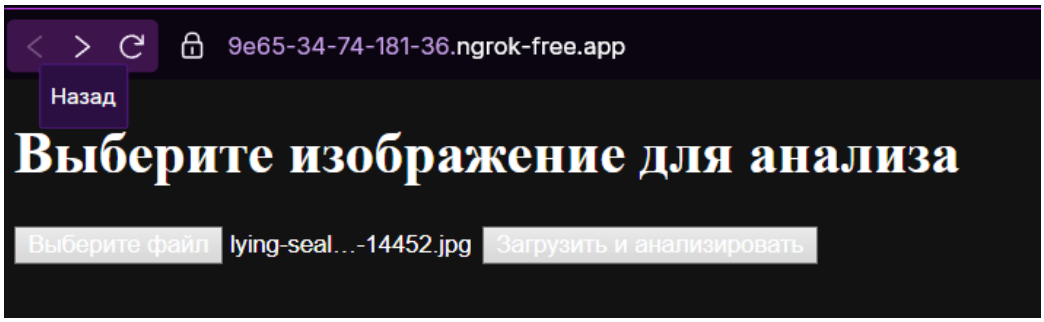


Рисунок 24 – Загрузка фотографии для тестирования

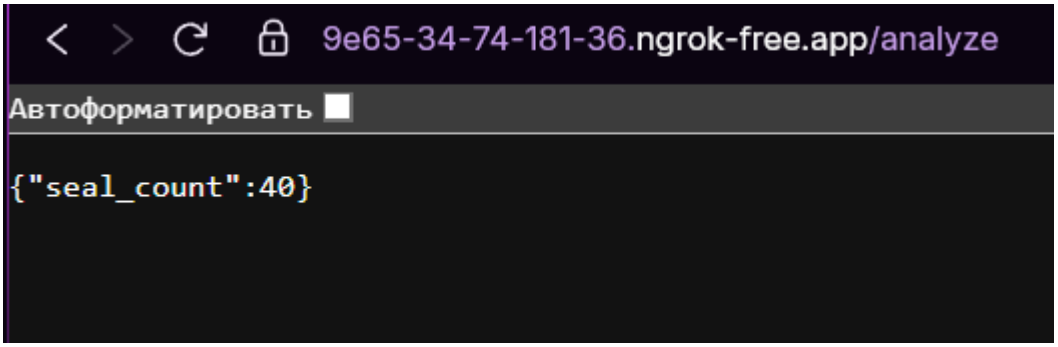


Рисунок 25 – Результат тестирования обученной модели

Выбор фотографий на эмуляторе приложения представлен на рисунке 26.

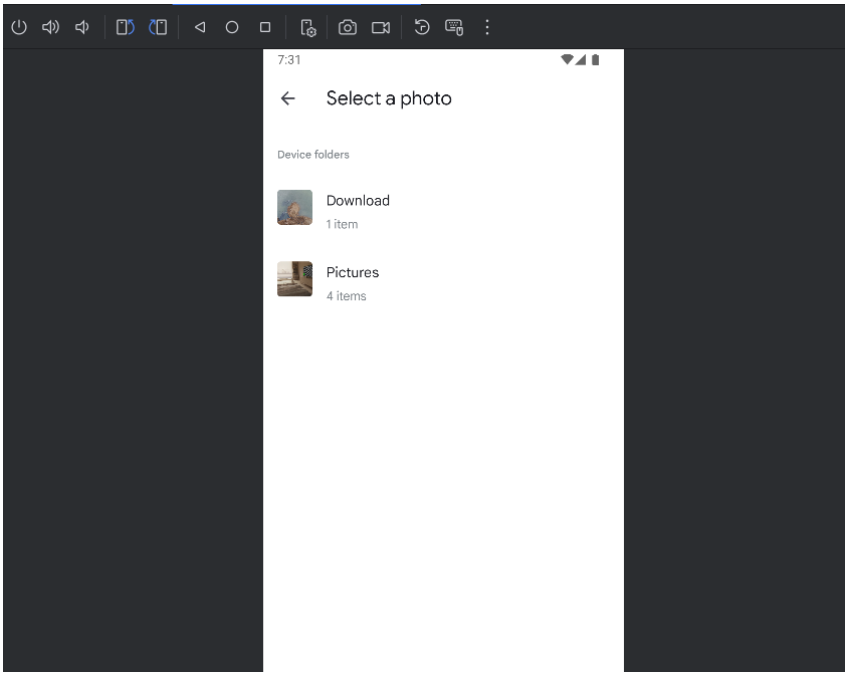


Рисунок 26 – Выбор фотографий в эмуляторе

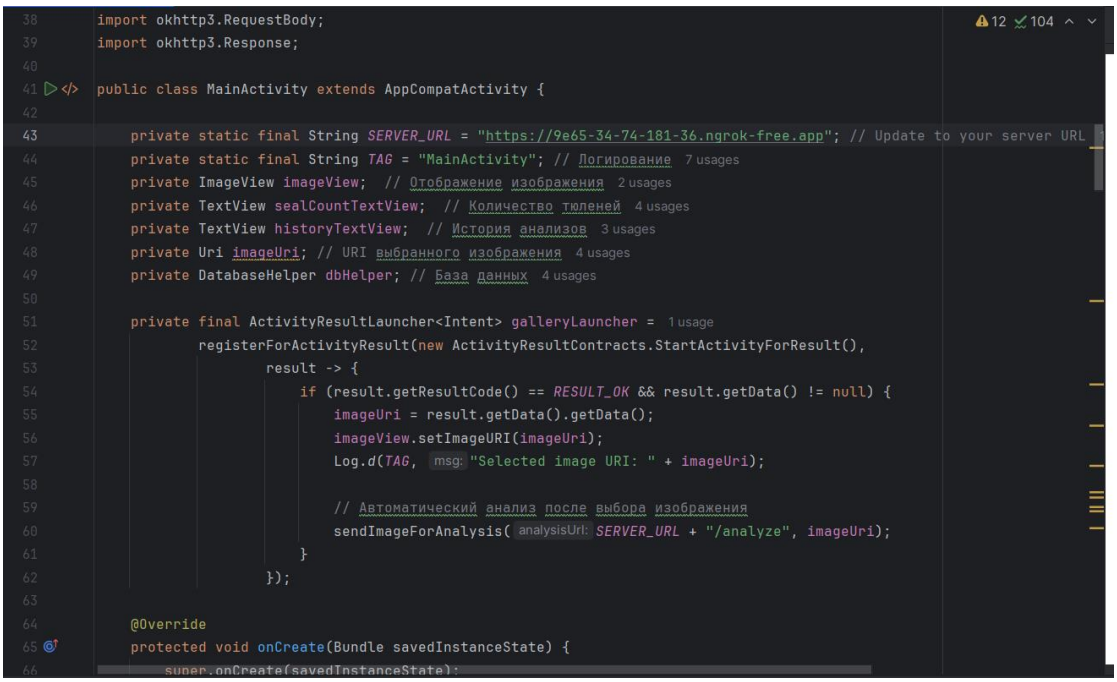
3.2 Тестирование ИС

После завершения разработки было проведено тестирование функционала с целью выявления ошибок и проверки точности работы алгоритмов.

Этапы тестирования:

Модульное тестирование.

Проверили работу отдельных компонентов приложения (базы данных, загрузки изображений, взаимодействия с сервером) (рисунки 27-34). Также были добавлены кнопки, чтобы можно было просматривать историю базы данных картин, и очищать ее чтобы не загружать память.



```
import okhttp3.RequestBody;
import okhttp3.Response;

public class MainActivity extends AppCompatActivity {

    private static final String SERVER_URL = "https://9e65-34-74-181-36.ngrok-free.app"; // Update to your server URL
    private static final String TAG = "MainActivity"; // Логирование  7 usages
    private ImageView imageView;  // Отображение изображения  2 usages
    private TextView sealCountTextView;  // Количество тюленей  4 usages
    private TextView historyTextView;  // История анализов  3 usages
    private Uri imageUri; // URI выбранного изображения  4 usages
    private DatabaseHelper dbHelper; // База данных  4 usages

    private final ActivityResultLauncher<Intent> galleryLauncher =  1 usage
            registerForActivityResult(new ActivityResultContracts.StartActivityForResult(),
                    result -> {
                        if (result.getResultCode() == RESULT_OK && result.getData() != null) {
                            imageUri = result.getData().getData();
                            imageView.setImageURI(imageUri);
                            Log.d(TAG,  msg: "Selected image URI: " + imageUri);

                            // Автоматический анализ после выбора изображения
                            sendImageForAnalysis( analysisUrl: SERVER_URL + "/analyze", imageUri);
                        }
                    });

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

Рисунок 27 – Код начала главного файла приложения

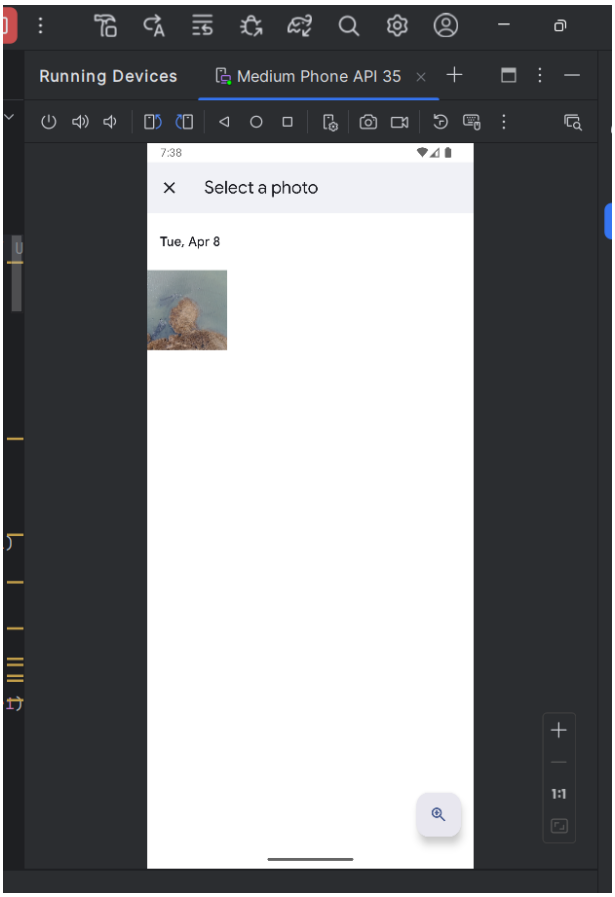


Рисунок 28 – Выбор фотографий на эмуляторе

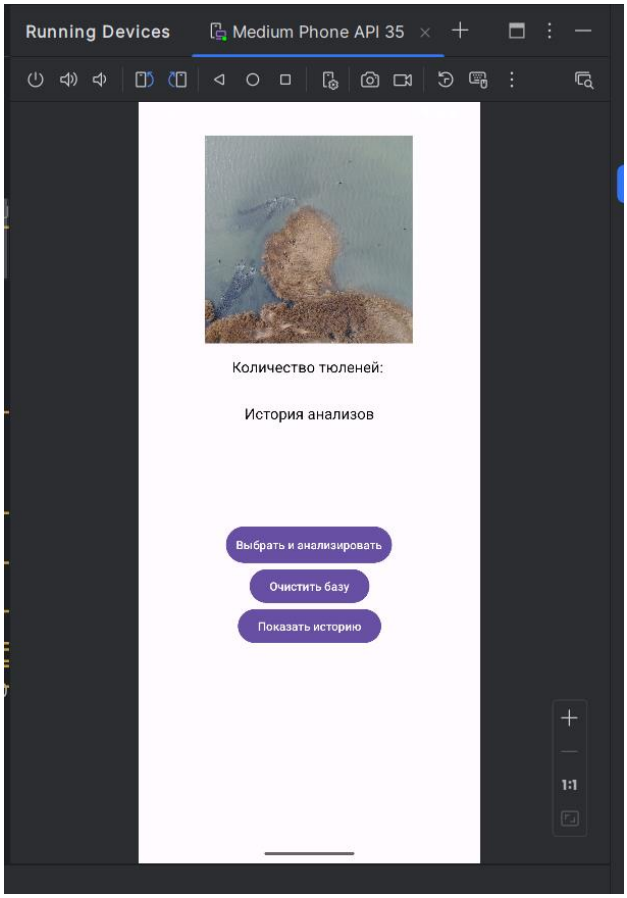


Рисунок 29 - Ожидание результата сервера

Рисунок 30 – Результат пришедший с сервера при анализе фотографии

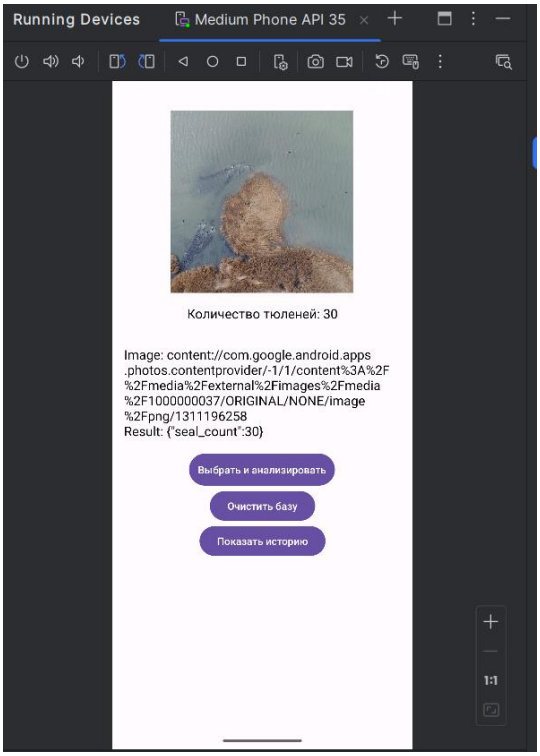


Рисунок 31 - Демонстрация работы кнопки показывающая историю базы данных анализов всех фотографий

Рисунок 32 - Фотография для анализа

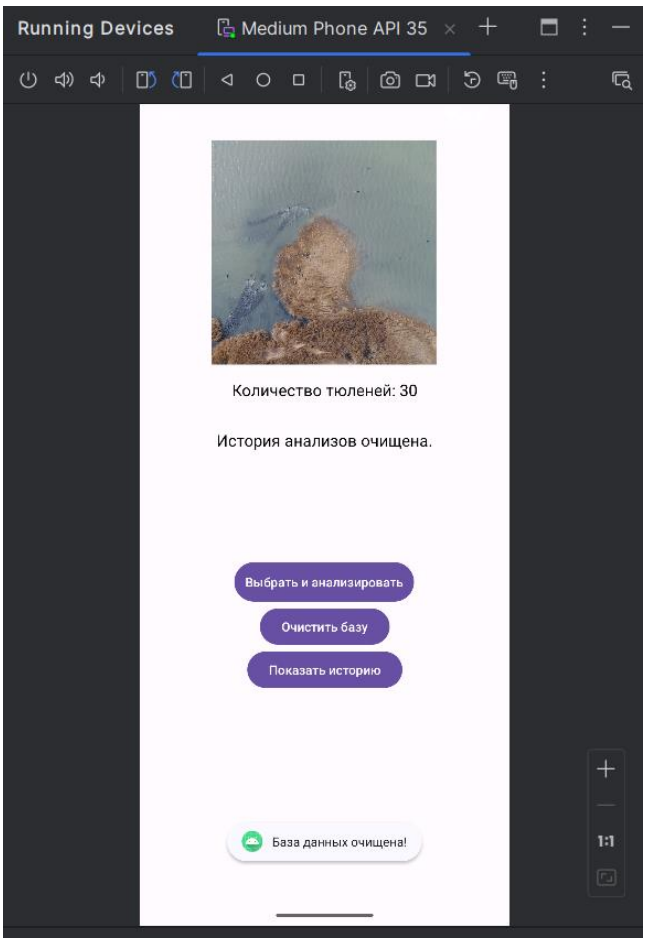


Рисунок 33 – Демонстрация работы кнопки очищение истории базы данных

```
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.28.0.12:5000
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug:127.0.0.1 - - [13/May/2025 07:28:44] "GET / HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [13/May/2025 07:28:45] "GET /favicon.ico HTTP/1.1" 404 -

image 1/1 /content/drive/MyDrive/dataset/train/images/temp_47cf9d91-057c-45f2-b822-ed7a18b71ce2.jpg: 448x640 43 seals, 1788.7ms
Speed: 19.2ms preprocess, 1788.7ms inference, 54.7ms postprocess per image at shape (1, 3, 448, 640)
INFO:werkzeug:127.0.0.1 - - [13/May/2025 07:29:07] "POST /analyze HTTP/1.1" 200 -

image 1/1 /content/drive/MyDrive/dataset/train/images/temp_44bd8326-b987-4a2f-8b95-e7878b27d943.jpg: 384x640 32 seals, 1547.9ms
Speed: 5.0ms preprocess, 1547.9ms inference, 1.6ms postprocess per image at shape (1, 3, 384, 640)
INFO:werkzeug:127.0.0.1 - - [13/May/2025 07:37:27] "POST /analyze HTTP/1.1" 200 -
```

Рисунок 34 - Демонстрация взаимодействия эмулятора с сервером

Интеграционное тестирование:

Проверили взаимодействие всех частей системы, включая связь с облачными сервисами (рисунок 35). Ошибок как на предыдущем скриншоте не было выявлено.

```
        client.newCall(request).enqueue(new Callback() {
            @Override
            public void onFailure(Call call, IOException e) {
                Log.e(TAG, msg: "Ошибка отправки изображения: " + e.getMessage(), e);
                runOnUiThread(() -> sealCountTextView.setText("Ошибка анализа: " + e.getMessage()));
            }

            @Override  4 usages
            public void onResponse(Call call, Response response) throws IOException {
                if (response.isSuccessful()) {
                    String responseBody = response.body().string();
                    Log.d(TAG,  msg: "Ответ сервера: " + responseBody);

                    // Извлекаем количество тюленей из ответа
                    String sealCount = extractSealCount(responseBody);

                    // Обновляем только `sealCountTextView`
                    runOnUiThread(() -> sealCountTextView.setText("Количество тюленей: " + sealCount));

                    // Сохранение результата в базу данных
                    dbHelper.insertAnalysisResult(imageUri.toString(), responseBody);
                } else {
                    Log.e(TAG,  msg: "Ошибка сервера: " + response.message());
                }
            }
        });
    }
```

Рисунок 35 - Выявления ошибок

Функциональное тестирование:

Провели тесты на реальных изображениях тюленей, проверяя корректность распознавания объектов (рисунки 36-38).

Рисунок 36 - Фотография для анализа

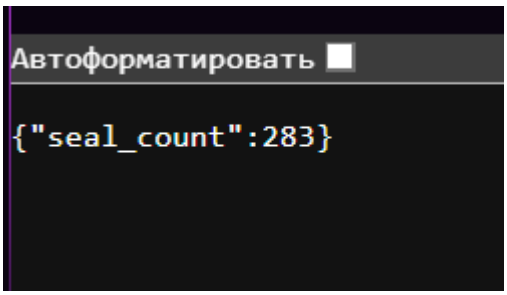


Рисунок 37 - Результат анализа фотографии

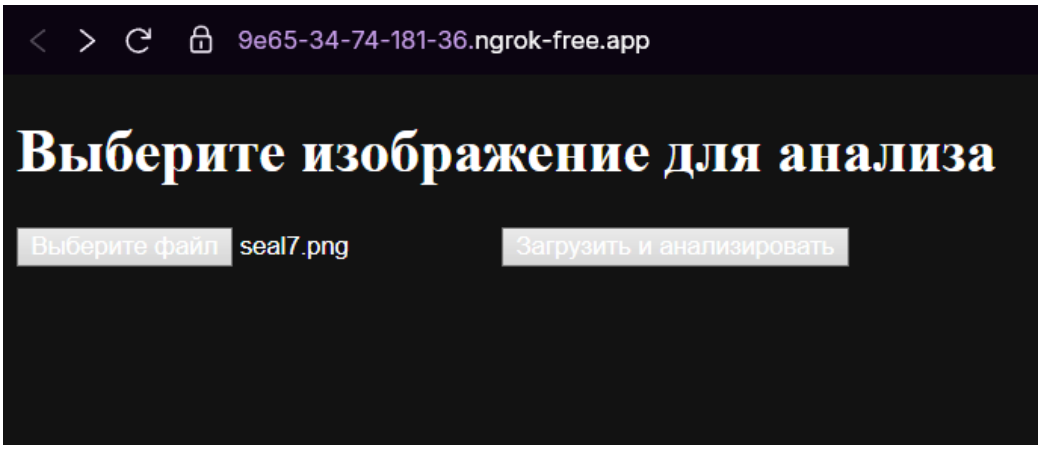


Рисунок 38- Показ страницы сайта с встроенным анализом фотографий

Такой же результат был и в эмуляторе приложения.

Оптимизация:

Улучшили скорость обработки изображений и взаимодействие с сервером. При обучении модели в коле видно, что применено больше epoch для лучшего обучения модели

```
#    Установка необходимых библиотек
!pip install --upgrade ultralytics torch torchvision torchaudio

# □ Импорт библиотек
from google.colab import drive
from ultralytics import YOLO
from PIL import Image, ImageEnhance
import cv2  # Улучшенная обработка изображений
import torch
import numpy as np
import os
import shutil
import random
import gc  # Автоочистка памяти
```

```
# □ Подключение Google Диска
drive.mount('/content/drive', force_remount=True)

# □ Проверяем доступность GPU
device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"□ Используем устройство: {device}")

# □ Функция аугментации изображений (улучшено качество и детализация)
def augment_images(input_folder, output_folder):
    os.makedirs(output_folder, exist_ok=True)

    for img_name in os.listdir(input_folder):
        img_path = os.path.join(input_folder, img_name)
        img = Image.open(img_path).convert("RGB")

        # □ Увеличение контрастности
        enhancer = ImageEnhance.Contrast(img)
        img = enhancer.enhance(2.0)

        # □ Фильтрация шума
        img_np = cv2.cvtColor(np.array(img), cv2.COLOR_RGB2BGR)
        img_np = cv2.GaussianBlur(img_np, (3, 3), 0)
        img = Image.fromarray(cv2.cvtColor(img_np, cv2.COLOR_BGR2RGB))

        # □ Оптимальный размер для YOLOv8
        img = img.resize((640, 640))

        # Вращение
        img_rotate = img.rotate(15)
        img_rotate.save(os.path.join(output_folder, f"rotate_{img_name}"), format="JPEG")

        # Отражение
        img_flip = img.transpose(Image.FLIP_LEFT_RIGHT)
        img_flip.save(os.path.join(output_folder, f"flip_{img_name}"), format="JPEG")

        # Яркость
        enhancer = ImageEnhance.Brightness(img)
        img_bright = enhancer.enhance(1.2)
        img_bright.save(os.path.join(output_folder, f"bright_{img_name}"), format="JPEG")

        #    Очистка памяти
        del img
        gc.collect()

    print("□ Аугментация завершена! Датасет расширен.")

# □ Автоматическое разделение изображений на train/val (проверка `.txt` меток!)
def split_dataset(dataset_path):
```

```
    images_path = os.path.join(dataset_path, "train/images")
    labels_path = os.path.join(dataset_path, "train/labels")
    val_images_path = os.path.join(dataset_path, "val/images")
    val_labels_path = os.path.join(dataset_path, "val/labels")

    os.makedirs(val_images_path, exist_ok=True)
    os.makedirs(val_labels_path, exist_ok=True)

    image_files = sorted([f for f in os.listdir(images_path) if f.endswith(".jpg")])

    val_count = int(len(image_files) * 0.3)
    val_samples = random.sample(image_files, val_count)

    for img_file in val_samples:
        src_img = os.path.join(images_path, img_file)
        dest_img = os.path.join(val_images_path, img_file)

        label_file = img_file.replace(".jpg", ".txt")
        src_label = os.path.join(labels_path, label_file)
        dest_label = os.path.join(val_labels_path, label_file)

        # □ Проверяем, существует ли `.txt` файл перед перемещением!
        if os.path.exists(src_label):
            shutil.move(src_img, dest_img)
            shutil.move(src_label, dest_label)
        else:
            print(f"□ Пропущено: метка {src_label} не найдена!")

    print(f"□ Перемещено {val_count} изображений в папку валидации.")

# □ Запуск аугментации перед обучением
augment_images("/content/drive/MyDrive/dataset/train/images",
"/content/drive/MyDrive/dataset/train/images_aug")

# □ Запуск разделения данных
split_dataset("/content/drive/MyDrive/dataset")

# □ Инициализация предобученной YOLOv8 (обработка ошибок!)
try:
    model = YOLO("yolov8m.pt")  # Используем среднюю версию YOLO для лучшего качества
except Exception as e:
    raise RuntimeError(f"□ Ошибка загрузки модели: {e}")

# □ Проверка `data.yaml`, содержит ли `"seal"`
with open("/content/drive/MyDrive/dataset/data.yaml", "r") as f:
    data_yaml = f.read()
if "seal" not in data_yaml:
    raise ValueError("□ Ошибка: `seal` не найден в data.yaml! Проверьте разметку.")
```

```
# 🚀 Запуск обучения модели YOLOv8 (улучшено качество)
model.train(
   data="/content/drive/MyDrive/dataset/data.yaml",
   epochs=300,  # Длительное обучение для высокой точности
   imgsz=640,
   batch=4,  # Оптимизировано для Colab (снижение нагрузки)
   optimizer="AdamW",  # Улучшенный Adam для стабильного обучения
   device=device
)

# 📂 **Правильное сохранение модели**
model_path = "/content/runs/detect/train/weights/best.pt"
model.save(model_path)  # Гарантированно сохраняет YOLO

# 📂 Перемещение файла в Google Drive
drive_model_path = "/content/drive/MyDrive/seal_detector.pt"

if os.path.exists(model_path):
   print("📂 Файл найден, перемещаем в Google Drive...")
   os.makedirs("/content/drive/MyDrive", exist_ok=True)
   shutil.move(model_path, drive_model_path)
   print(f"✅ Модель успешно сохранена в Google Drive: {drive_model_path}")
else:
   print("❌ Ошибка: файл модели не найден! Проверьте путь и попробуйте снова!")
```

3.3 Развертывание ИС

После тестирования приложение было подготовлено к развертыванию и возможному масштабированию. Как показано при тестирование проблем выявлений ошибок у кода не возникло (рисунок 39).

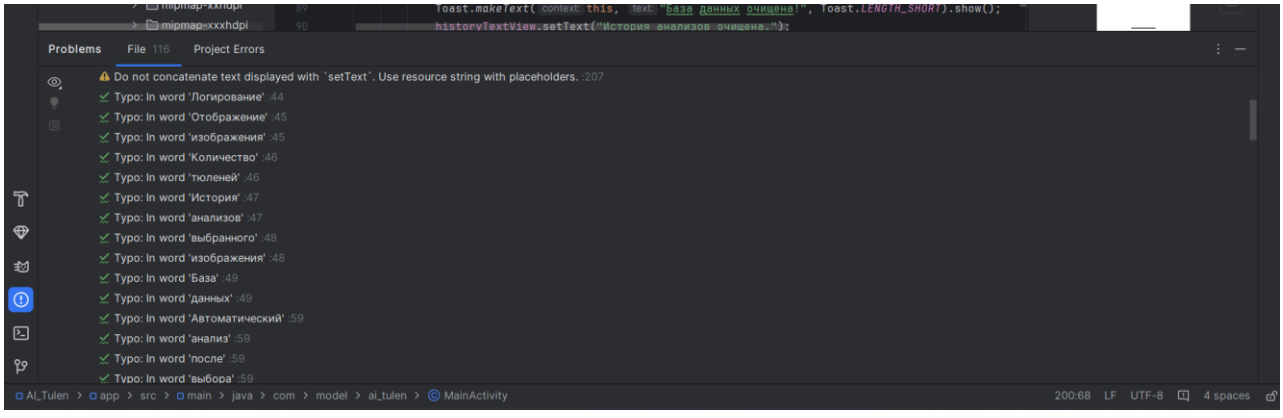


Рисунок 39 - Выявления ошибок в терминале приложения

ЗАКЛЮЧЕНИЕ

Разработка мобильного приложения **AI_TULEN** прошла полный цикл, начиная от формулирования требований и проектирования архитектуры, до тестирования, оптимизации и развертывания. В процессе разработки были применены современные технологии, обеспечивающие точность анализа изображений, надежность обработки данных и удобство использования.

Одним из ключевых аспектов разработки стало использование Make Sense для разметки изображений тюленей, что позволило создать качественный датасет для машинного обучения. Экспорт данных в YOLO обеспечил интеграцию с нейросетевыми моделями, а OpenCV помог обработать изображения с высокой точностью.

Для хранения данных использовалась SQLite, что позволило организовать локальное хранилище истории анализов, а интеграция с Google Cloud обеспечила возможность масштабируемой обработки изображений в облаке. Взаимодействие между клиентом и сервером было реализовано через OkHttp, что позволило быстро и надежно передавать данные.

После проведения тестирования, были оптимизированы ключевые процессы, включая работу с изображениями, обмен данными и обработку результатов анализа. В результате приложение стало интуитивно понятным для пользователей, стабильным в работе и готовым к дальнейшему развитию.

Используемые технологии обеспечили производительность, точность детекции объектов и надежность хранения данных. Проект **AI_TULEN** готов к расширению функционала, внедрению новых алгоритмов анализа изображений, а также интеграции дополнительных возможностей для исследования тюленей. Но также стоит отметить, что для дальнейшего расширения функционала нужно больше данных при обучения модели, для этого стоит больше уделить внимание экологии и жизни тюленей

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1 Troelsen A., Japikse Ph. Pro C# 10 with .NET 6: Foundational Principles and Practices in Programming. Apress Media, LLC. 2022. - 1680 с.

2 Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley books. - 431 с.

3 Мартин Р. Чистый код: создание, анализ и рефакторинг. Библиотека программиста. Издательство Питер. 2022. – 464 с.

4 Martin R. Design Principles and Design Patterns. 2000. objectmentor.com. Archived from the original on 2015-09-06.

5 Andersson J. Learning Microsoft Azure: Cloud Computing and Development Fundamentals. O'Reilly Media. 2023. – 480 с.

6 Microsoft. The TypeScript Handbook.// Электронная версия на сайте https://www.typescriptlang.org/docs/handbook/intro.html

7 GitHub. GitHub Actions documentation.// Электронная версия на сайте https://docs.github.com/en/actions

8 Roth D., Anderson R., Luttin Sh. Overview of ASP.NET Core.// Электронная версия на сайте https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-9.0

9 Language Integrated Query (LINQ) .// Электронная версия на сайте https://learn.microsoft.com/en-us/dotnet/csharp/linq/

10 Goodfellow I., Bengio Y., Courville A. Deep Learning. Cambridge, MA: MIT Press, 2016. — 800 с.

11 Бишоп К.М.. Нейронные сети для распознавания образов (перевод с англ.). М.: Мир, 2000. — 504 с.

12 Bishop Chr.M. Pattern Recognition and Machine Learning. Springer, 2006. — 738 с.

13 Haykin S. Neural Networks and Learning Machines (3-е изд.). Pearson Education, 2009. — 936 с.

14 LeCun Y., Bengio Y., Hinton G. Deep learning. Nature, vol. 521 (7553), 2015, С. 436–444.

15 Russakovsky O. et al. ImageNet Large Scale Visual Recognition Challenge. Int. J. Comput. Vision, vol. 115, 2015, С. 211–252.

16 Krizhevsky A., Sutskever I., Hinton G.. ImageNet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing Systems, vol. 25, 2012, С. 1097–1105.

17 Szegedy C. et al. Going Deeper with Convolutions. Proc. IEEE CVPR 2015, С.1-9.

18 He K. et al. Deep Residual Learning for Image Recognition. Proc. IEEE CVPR 2016, С. 770–778.

19 Ren S. et al. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. Adv. in Neural Information Processing Systems, vol. 28 (NeurIPS 2015), С. 91–99.

20 Lin T.-Y. et al. Feature Pyramid Networks for Object Detection. Proc. IEEE CVPR 2017, C. 2117–2125.

21 He K. et al. Mask R-CNN. Proc. IEEE ICCV 2017, C. 2980–2988.

22 Redmon J. et al. You Only Look Once: Unified, Real-Time Object Detection. Proc. IEEE CVPR 2016, C. 779–788.

23 Long J., Shelhamer E., Darrell T. Fully Convolutional Networks for Semantic Segmentation. Proc. IEEE CVPR 2015, C. 3431–3440.

24 Simonyan K., Zisserman A.. Very Deep Convolutional Networks for Large-Scale Image Recognition. ICLR 2015 (препринт, arXiv:1409.1556), 16 с.

25 Ng A., LeCun Y. (ред.). Deep Learning: примеры, интерфейсы и библиотеки. (отчёт Stanford/NYU), 2017. — 80 с.

Приложение А

Техническое задание

Техническое задание на разработку информационной системы для помощи экологам в информировании о близлежащих каспийских тюленях

Создать мобильное приложение с встроенной информационной системой, использующее нейросеть для анализа изображений каспийских тюленей, снятых с дронов. Приложение должно предоставлять экологам данные о местоположении, количестве тюленей и их активности, а также визуализировать результаты анализа в удобном формате.

А.1 Основание для разработки

Система разрабатывается в рамках научного проекта ИРН BR18574148 «Развитие экологических систем и мониторинга объектов окружающей среды».

А.2 Назначение

Данная система разработана с целью автоматизации мониторинга каспийских тюленей посредством анализа изображений, снятых с высоты дронами. Она предоставляет экологам данные о количестве и местоположении тюленей, визуализирует эти данные, а также сохраняет их для последующего анализа. Приложение облегчает сбор данных о популяции тюленей и позволяет оперативно реагировать на изменения в экосистеме.

А.3 Требования к функциональным характеристикам

Система должна обеспечить выполнение следующих функций:

– **Распознавание объектов**:
  - Анализ изображений с помощью встроенной нейросети.
  - Определение тюленей и выделение их bounding boxes на изображении.

– **Подсчёт объектов**:
  - Подсчёт количества тюленей на фотографии и отображение результата.

– **Хранение данных**:
  - Сохранение результатов обработки в локальную базу данных.
  - Возможность доступа к истории результатов.

– **Удобный пользовательский интерфейс**:
  - Возможность загрузки изображений через приложение.

- Отображение результатов в виде изображения с выделенными объектами.

– **Выгрузка данных**:
  - Экспорт данных о популяции тюленей в формате отчёта (PDF или Excel).

А.4 Требования к надежности

Система должна обеспечивать высокую стабильность работы даже при обработке больших объёмов данных.

Надёжная работа в офлайн-режиме (с сохранением данных в локальной базе данных).

Обеспечение резервного копирования данных для предотвращения потери информации.

Обеспечение корректной работы при низком уровне освещения на фотографиях (учёт характеристик изображений при обучении модели).

А.5 Требованию к составу и параметрам технических средств

Мобильные устройства:

- Операционная система Android версии 6.0 и выше.
- Минимальные характеристики:
  - ОЗУ: 2 ГБ.
  - Процессор: ARM64.
  - Свободное место: 100 МБ для хранения изображений и данных.

Разработческое ПО:

- Android Studio для создания пользовательского интерфейса и интеграции нейросети.
- Google Colab для обучения модели и её конвертации в TensorFlow Lite.

Библиотеки:

- TensorFlow Lite для инференса модели на устройстве.
- SQLite для хранения данных локально.
- Google Maps API для отображения географической информации.

Приложение Б

1. Код для загрузки и обработки изображения

```
import android.content.Intent;
import android.graphics.Bitmap;
import android.os.Bundle;
import android.provider.MediaStore;
import android.widget.ImageView;
import android.widget.TextView;

import androidx.appcompat.app.AppCompatActivity;

import org.tensorflow.lite.Interpreter;

import java.nio.ByteBuffer;

public class MainActivity extends AppCompatActivity {
    private Interpreter tflite;
    private ImageView imageView;
    private TextView resultText;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        imageView = findViewById(R.id.imageView);
        resultText = findViewById(R.id.resultText);

        findViewById(R.id.buttonLoadImage).setOnClickListener(v -> {
            Intent intent = new Intent(Intent.ACTION_PICK,
MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
            startActivityForResult(intent, 1);
        });
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if (requestCode == 1 && resultCode == RESULT_OK && data != null) {
            try {
                Bitmap bitmap =
MediaStore.Images.Media.getBitmap(this.getContentResolver(), data.getData());
                imageView.setImageBitmap(bitmap);
```

```
            // Предобработка изображения
            ByteBuffer inputImage = preprocessImage(bitmap);

            // Выполнение инференса модели
            float[][] output = detectObjects(inputImage);
            resultText.setText("Обнаружено тюленей: " + output.length);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

private ByteBuffer preprocessImage(Bitmap bitmap) {
    // Пример предобработки изображения для подачи в TensorFlow Lite
    // Изменение размера и нормализация
    bitmap = Bitmap.createScaledBitmap(bitmap, 320, 320, true);
    ByteBuffer buffer = ByteBuffer.allocateDirect(4 * 320 * 320 * 3); // Размер
320x320 с 3 каналами (RGB)
    buffer.rewind();
    for (int y = 0; y < 320; y++) {
        for (int x = 0; x < 320; x++) {
            int pixel = bitmap.getPixel(x, y);
            buffer.putFloat(((pixel >> 16) & 0xFF) / 255.0f); // Красный канал
            buffer.putFloat(((pixel >> 8) & 0xFF) / 255.0f);  // Зелёный канал
            buffer.putFloat((pixel & 0xFF) / 255.0f);         // Синий канал
        }
    }
    return buffer;
}

private float[][] detectObjects(ByteBuffer inputImage) {
    float[][] output = new float[10][4]; // Максимум 10 объектов с рамками
    tflite.run(inputImage, output);
    return output;
}
}
```

Продолжение приложания Б

2. Код для работы с Google Maps API

```
javascript
function initializeMap(latitude, longitude) {
    const center = { lat: latitude, lng: longitude };
    const map = new google.maps.Map(document.getElementById("map"), {
        center: center,
        zoom: 10,
    });

    new google.maps.Marker({
        position: center,
        map: map,
        title: "Стадо тюленей здесь!",
    });
}

// Пример вызова:
initializeMap(47.1100, 51.9200); // Координаты местоположения
```