**МИНОБРНАУКИ РОССИИ**

**Федеральное государственное автономное образовательное**

**учреждение высшего образования**

**«Южный федеральный университет»**

**Институт высоких технологий и пьезотехники**

**Кафедра прикладной информатики и инноватики**

**Рубашевская Анастасия Андреевна**

**Приложение Telegram-бот для распознавания достопримечательностей Ростова**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ**

**РАБОТА БАКАЛАВРА**

**по направлению 09.03.03 – Прикладная информатика**

**Научный руководитель –**

**Старший преподаватель Яценко Д. В.**

**Ростов-на-Дону** – **2024**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ**

**ИНСТИТУТ ВЫСОКИХ ТЕХНОЛОГИЙ И ПЬЕЗОТЕХНИКИ**

**Кафедра прикладной информатики и инноватики**

**З А Д А Н И Е**

**на выпускную квалификационную работу**

**Студент** гр. 4–9 Рубашевская А. А.

1. **Тема:** Приложение Telegram-бот для распознавания достопримечательностей Ростова
2. **Срок сдачи законченной работы** \_\_.\_\_.2025
3. **Исходные данные:**

Государственные стандарты

1. ГОСТ 7.1-2003 «Библиографическая запись. Библиографическое описание. Общие требования и правила составления»;

2. ГОСТ 7.82-2001 «Библиографическая запись. Библиографическое описание электронных ресурсов»

3. ГОСТ 19.001-77. Общие положения;

4. ГОСТ 19.781-90. Термины и определения;

5. ГОСТ 19.701-90 (ИСО 5807-85). Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения;

6. ГОСТ 19.103-77. Обозначения программ и программных документов;

7. ГОСТ 19.101-77. Виды программ и программных документов;

8. ГОСТ 19.106-78. Требования к программным документам, выполненным печатным способом;

9. ГОСТ 19.104-78. Основные надписи;

10. ГОСТ 19.301-79. Программа и методика испытаний. Требования к содержанию и оформлению;

11. ГОСТ 19.402-78. Описание программы;

12. 58 13. ГОСТ 19.401-78. Текст программы. Требования к содержанию и оформлению;

14. ГОСТ 19.504-79. Руководство программиста. Требования к содержанию и оформлению;

15. ГОСТ 19.505-79. Руководство оператора. Требования к содержанию и оформлению;

16. ГОСТ 19.202-78. Спецификация. Требования к содержанию и оформлению.

**4. Перечень вопросов, подлежащих разработке:**

1. Анализ предметной области.
2. Обоснование технических решений.
3. Проектирование элементов системы.
4. Реализация компонентов системы.
5. Разработка методики тестирования.
6. **Перечень графического материала:**

Подготовка графических материалов для презентации работы

1. **Консультанты по работе:**
2. **Дата выдачи задания: 23.09.2024**
3. **Руководитель Яценко Д. В.**

*Подпись ФИО*

1. **Задание принято к исполнению**

*Дата Подпись студента*

АННОТАЦИЯ

В ходе выполнения выпускной квалификационной работы была сформулирована первичная постановка задачи. Затем были исследованы аналогичные продукты. Среди существующих, предлагающих функцию распознавания достопримечательностей были выбраны наиболее популярные, в процессе исследования были выделены сильные и слабые стороны. На основе этих данных были сформированы требования к функционалу в разрабатываемой системе.

В работе изложен процесс работы над приложением, а также аргументированы выборы тех или иных технологий и инструментов для разработки системы.

Было принято решение разрабатывать приложение Telegram-бот. Язык программирования, выбранный для разработки – Python. Нейронная сеть, отвечающая за распознавание достопримечательностей, была реализована с использованием PyTorch. Для создания пользовательского интерфейса и функционала для взаимодействия с пользователем была использована библиотека Telebot, обеспечивающая удобное взаимодействие с Telegram.

Важной частью работы над продуктом являлось сбор и подготовка датасета фотографий достопримечательностей г. Ростова-на-Дону.

Результатом работы является приложение, реализующее распознавание достопримечательностей при помощи искусственного интеллекта, который основывается на данных о достопримечательностях Ростова.

Ключевые слова: достопримечательность, Telegram-бот, нейронные сети, датасет, распознавание изображений, Python, PyTorch, Telebot.

ANNOTATION

During the completion of the final qualification work, the primary statement of the task was formulated. Then similar products were investigated. Among the existing applications offering the functionality of recognition of sights, the most popular ones were selected, and strengths and weaknesses were highlighted during the research. Based on these data, the requirements for the functionality in the system under development were formed.

The paper describes the process of working on the application, as well as the reasoned choices of certain technologies and tools for system development.

It was decided to develop an application Telegram bot for. The programming language chosen for development is Python. The neural network responsible for recognizing landmarks was implemented using PyTorch. To create a user interface and functionality for user interaction, the Telebot library was used, which provides convenient interaction with Telegram.

An important part of the work on the product was the collection and preparation of a dataset of photographs of the sights of Rostov-on-Don.

The result of the work is an application that implements recognition of landmarks using artificial intelligence, which is based on data about the sights of Rostov.

Keywords: attraction, Telegram bot, neural networks, dataset, image recognition, Python, PyTorch, Telebot.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 8](#_Toc194442196)

[1. ПОСТАНОВКА ЗАДАЧИ 10](#_Toc194442197)

[1.1 Теоретические основы распознавания изображений 10](#_Toc194442198)

[1.2 Светочные нейронные сети и их применение в задачах компьютерного зрения 12](#_Toc194442199)

[1.3 Первичная постановка задачи 14](#_Toc194442200)

[1.4 Декомпозиция задачи 14](#_Toc194442201)

[1.5 Анализ существующих аналогов 15](#_Toc194442202)

[1.6 Функциональные требования 18](#_Toc194442203)

[1.7 Обоснование выбора платформы приложения 18](#_Toc194442204)

[1.7.1 Выбор языка программирования 19](#_Toc194442205)

[1.7.2 Выбор среды разработки приложения 23](#_Toc194442206)

[1.8 Выводы и результаты по главе 23](#_Toc194442207)

[2. ПРОЕКТИРОВАНИЕ СИСТЕМЫ 24](#_Toc194442208)

[2.1 Проектирование системы, UML моделирование 24](#_Toc194442209)

[2.1.1 Диаграммы вариантов использования 25](#_Toc194442210)

[2.1.2. Диаграмма классов 31](#_Toc194442211)

[2.3 Проектирование интерфейса приложения 33](#_Toc194442212)

[2.4 Выводы и результаты по главе 37](#_Toc194442213)

[3. РЕАЛИЗАЦИЯ СИСТЕМЫ 38](#_Toc194442214)

[3.1 Реализация нейронной сети 38](#_Toc194442215)

[3.1.1 Создание набора данных 40](#_Toc194442216)

[3.1.2 Формирование логики нейронной сети 46](#_Toc194442217)

[3.2 Реализация Telegram-бота 53](#_Toc194442218)

[3.3 Выводы и результаты по главе 57](#_Toc194442219)

[4. БЕЗОПАСНОСТЬ ЖИЗНЕДЕЯТЕЛЬНОСТИ 58](#_Toc194442220)

[5. ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ 61](#_Toc194442221)

[6. 1 Определение стоимости разработки 61](#_Toc194442222)

[6. 2 Экономическая выгода разработки 62](#_Toc194442223)

[ЗАКЛЮЧЕНИЕ 64](#_Toc194442224)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 65](#_Toc194442225)

[ПРИЛОЖЕНИЕ А 69](#_Toc194442226)

[ПРИЛОЖЕНИЕ Б 70](#_Toc194442227)

[ПРИЛОЖЕНИЕ В 73](#_Toc194442228)

# ВВЕДЕНИЕ

На сегодняшний день перед людьми часто встает задача понять, что изображено на фотографии, или какой объект находится перед ними. В такой ситуации на помощь приходят приложения, которые предоставляют функцию распознавания изображений. Зачастую, в основе таких приложений лежит концепция нейронных сетей.

Распознавание изображений – область искусственного интеллекта, в которой визуальные объекты обрабатываются и идентифицируются в попытке приблизить их максимально близко к человеческому пониманию.

Для классификации изображений ИИ использует нейронные сети, особенно сверточные нейронные сети (CNN), которые обучаются на больших наборах данных с маркированными изображениями [26].

В связи с быстрым развитием ИИ и машинного обучения, способность вычислительной техники понимать визуальный контент заметно улучшилась. Задача распознавания изображений может быть полезна во многих сферах нашей жизни: розничная торговля, здравоохранение, финансы, туризм и т.д.

Также распознавание изображений может быть полезно для пользователей социальных сетей. Для них самыми удобными и распространенными способами использования являются мобильные устройства, в частности смартфоны.

Смартфоны, в сравнении с персональными компьютерами, имеют ряд преимуществ, например, автономность, компактность и возможность решать разнообразные задачи. По этой причине, многие пользователи проводят достаточно много времени с мобильными устройствами. В рейтинге пяти самых популярных социальных сетей на момент 2024 года находится Телеграмм. Он предоставляет широкий и удобный пользователям функционал, а также возможность быстрого получения и обмена информацией.

Особую популярность сегодня имеют так называемые боты. Это специальные программы, которые способны выполнять конкретные функции в мессенджере, в том числе и распознавать изображения. Они также обеспечивают в Телеграмме удобство получения и обмена информацией. Уже существует большое количество ботов, которые способны распознавать печатный текст, изображения и т.д. Последнее и было выбрано в качестве проекта для разработки, так как такой продукт имеет высокую социальную значимость как для туристов, так и для местных жителей, а именно быстрый поиск информации о конкретной достопримечательности.

## 1. ПОСТАНОВКА ЗАДАЧИ

## 1.1 Теоретические основы распознавания изображений

На сегодняшний день научно-технический прогресс достиг невероятных результатов и продолжает стремительно прогрессировать. Благодаря этому, люди могут решать на повседневные задачи быстро и просто. Многие из них реализовываются с помощью искусственного интеллекта.

Искусственный интеллект (ИИ) – очень широкая область исследований, посвященная когнитивным способностям машин: обучение определенному поведению, упреждающее взаимодействие с окружающей средой, способность к логическому выводу и дедукции, компьютерное зрение, распознавание речи, решение задач, представление знаний, восприятие действительности и многое другое [25]. Самым популярным методом представления искусственного интеллекта является нейронные сети.

Нейронная сеть — это математическая модель и ее реализация в виде программной или программно-аппаратной реализации, которая основана на моделировании активности биологических нейронных сетей, которые представляют собой сети нейронов в биологическом организме [1].

Одна из самых популярных областей применения нейронных сетей – компьютерное зрение. Чтобы распознать изображение, нейронная сеть сначала должна быть обучена на данных. Это очень похоже на нейронные связи в человеческом мозгу — у нас есть некоторые знания, мы видим объект, анализируем его и идентифицируем [1].

Нейронные сети требовательны к набору данных, на котором они будут обучаться. Он должен быть подходящего размера, качественным и непротиворечивым. Такой набор данных можно скачать из различных источников, либо собрать самостоятельно.

Изображение разбито на небольшие участки, каждый из которых является входным нейроном. Между нейронами есть связи, которые называются синапсы. Синапсы хранят веса и различные коэффициенты (Рисунок 1.1). В процессе распознавания сигналы по синапсам передаются с одного уровня на другой, и нейроны сравнивают полученные сигналы с обработанными данными.

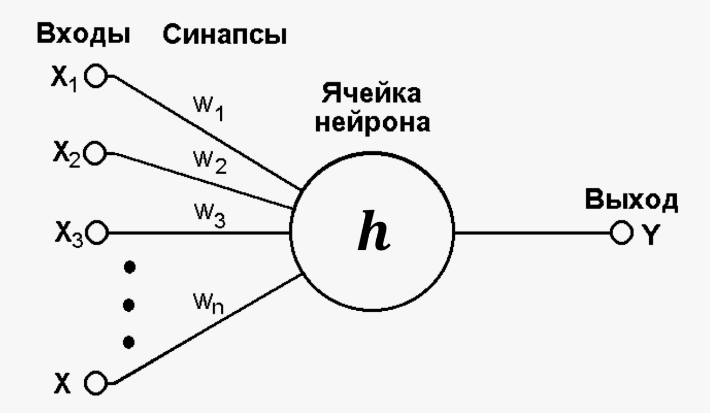


Рисунок 1.1 – Формальный нейрон

Вкратце, если мы попросим машину распознать фотографию кошки, мы разобьем фотографию на мелкие части и сравним эти слои с миллионами существующих изображений кошек — чертами, которые сеть усвоила [1].

Во время обучения нужно научить нейронную сеть определять нужное количество и значение характеристик для обеспечения хорошей точности на новых изображениях. При этом важно избегать переобучения, чтобы модель хорошо объясняла не только обучающий набор изображений, но и новые данные. После обучения нейронная сеть должна хорошо идентифицировать изображения, которые она не встречала в процессе обучения.

Также следует отметить, что в настоящее время существует большое количество стандартных архитектур для построения нейронных сетей, что значительно упрощает задачу построения нейронной сети с нуля и ограничивает ее выбором структуры сети, подходящей для конкретной задачи [1].

## 1.2 Светочные нейронные сети и их применение в задачах компьютерного зрения

Сверточные нейронные сети (convolutional neural networks, CNN) — это весьма широкий класс архитектур, основная идея которых состоит в том, чтобы переиспользовать одни и те же части нейронной сети для работы с разными маленькими, локальными участками входов. Как и многие другие нейронные архитектуры, сверточные сети известны довольно давно, и в наши дни у них уже нашлось много самых разнообразных применений, но основным приложением, ради которого люди когда-то придумали сверточные сети, остается обработка изображений [27].

Сверточные нейронные сети состоят из нескольких слоёв. Количество слоев определяет мощность архитектуры и способность обучаться. В состав сверточное нейронной сети могут входить следущие элементы:

● сверточный слой,

● пулинг,

● нормализация по бачу,

● полносвязный слой.

Сверточный слой позволяет выделить важные элементы изображения и удалить лишние. Это происходит с помощью ядер (фильтров), которые проходят по изображению и выполняют математические преобразования, которые позволяют учесть края, текстуры или более сложные детали объекта. Данная операция производится поэлементно, а результат объединяется в карту признаков.

Получение наиболее важной информации из карты признаков является основной задачей пулинга. Это достигается путем уменьшения изображений определенными методами. Наиболее распространенным из них является максимальный пулинг (max pooling), который выбираем максимальное значение в пределах одного окна (Рисунок 1.2). Такая операция позволяет избавиться от лишнего шума, а также при нахождении важного свойства опустить поиск других.

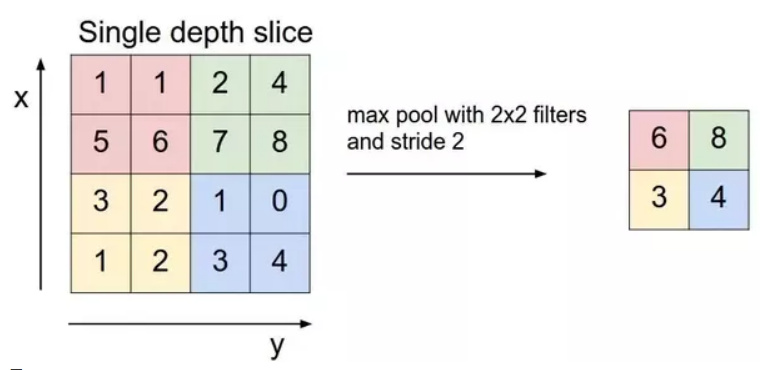


Рисунок 1. 2 - Пример пулинга максимального значения

Также существует метод, позволяющий повышать производительность и стабилизировать работу таких сложных архитектур, как нейронные сети. Этот метод называется нормализацией по батчу (Batch Normalization, BN). Он позволяет нормализовать признаки к одной шкале, что позволяет обеспечить равное влияние признаков на прогноз. Батч-нормализация позволяет ускорить настройку сети за счёт нормализации не только входных признаков, но и активаций промежуточного слоя. Слой батч-нормализации может включаться в любом месте сети - как за входным, так и за скрытым слоем [28].

Полносвязный слой служит для связи нейронов конкретного слоя с предыдущим, что позволяет определить «правильный ответ» на основе всех извлеченных признаков. Обычно полносвязные слои являются завершающими в нейронных сетях и служат непосредственно для классификации или регрессии.

Сложная архитектура, в частности сверточных нейронных сетей, делает их незаменимыми в области компьютерного зрения, при этом в широком круге задач. Все вышеупомянутые элементы и методики дают возможность эффективно извлекать признаки и делать прогноз с высокой точностью.

### 1.3 Первичная постановка задачи

Тема работы – приложение Telegram-бот для распознавания достопримечательностей Ростова.

Целью выпускной работы является создание программного продукта, помогающего людям понять, что за достопримечательность находится на фотографии. Для достижения цели необходимо изучить аналогичное программное обеспечение, отзывы и пожелания пользователей подобных продуктов. В результате создать конечный программный проект, совмещающий в себе возможности добавления пользователем фотографий, работы с искусственным интеллектом по вопросам распознавания изображений, а также простой интерфейс, тем самым давая пользователю удобный инструмент для идентификации достопримечательностей в городе Ростове-на-Дону.

## 1.4 Декомпозиция задачи

В ходе изучения рынка Telegram-ботов для распознавания объектов было выявлено, что многие из них имеют достаточно маленький набор токенов. Это является недостатком для пользователей, которые нуждаются в описании большого числа объектов. К сожалению, это не единственная отрицательная черта популярных аналогов, также некоторые сервисы предлагают пользователю выбрать нейронную сеть для распознавания объекта, что сильно усложняет работу. Также клиенту предлагается оформить платную подписку, чтобы увеличить количество запросов и функционал.

На смену этим сервисам должно прийти приложение Telegram-бот, которое будет иметь большее количество токенов, а также простой и удобный интерфейс. Данный программный продукт будет ориентирован конкретно на объекты Ростова-на-Дону, а его основная цель – распознать достопримечательность по фотографии пользователя.

Задача была разделена на следующие этапы:

1. Оценка существующих аналогов, анализ их недостатков и преимуществ;
2. Формулировка функциональных требований;
3. Спроектировать систему, которая будет соответствовать поставленным целям;
4. Разработать приложение Telegram-бот для распознавания достопримечательностей Ростова-на-Дону;
5. Протестировать полученное приложение и устранить недочеты;
6. Сделать конечный продукт активным.

### 1.5 Анализ существующих аналогов

На сегодняшний день уже существуют подобные приложения, которые способны распознавать изображения. Они также реализованы как Telegram-боты, имеющие свой функционал и интерфейс. Они предоставляют возможность пользователю добавить фотографию, а на выход получить ее описание. Один из разделов – информационный, который составлен на основе часто задаваемых вопросов, то есть инструкция по использованию бота. Однако функционал бота порой остается сложным для пользователя.

Отличительной особенностью разрабатываемого сервиса будет наличие простого для клиента функционала и удобного интерфейса. Telegram-бот будет способен распознавать и описывать объекты конкретно Ростова-на-Дону.

Один из популярных на данный момент аналогов – Gemini Google bot (Рисунок 1.3) содержит следующий функционал:

* Распознавание изображений и генерация описания конкретного объекта;
* Возможность оформить платную подписку;
* Возможность общения с искусственным интеллектом по вопросам пользователя – раздел «chat»;
* Возможность обратиться в техническую поддержку;
* Инструкция по пользованию ботом – раздел «info».

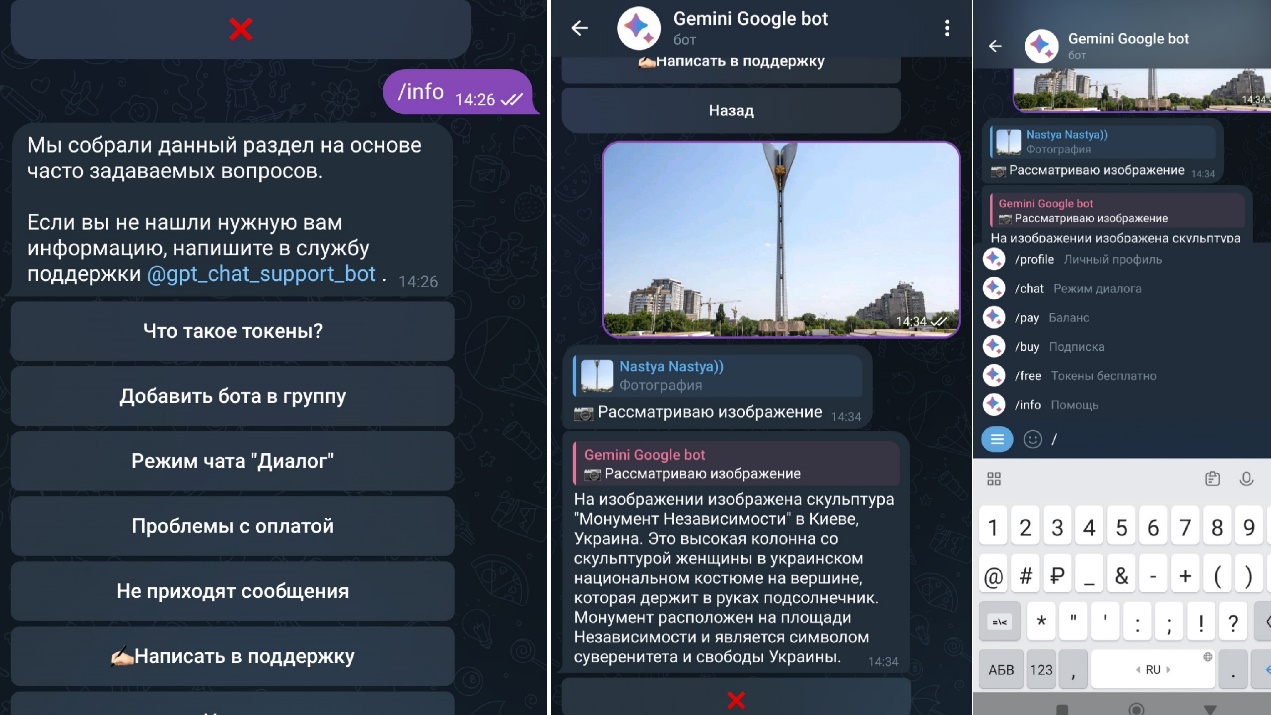


Рисунок 1.3 – Приложение Telegram-бот «Gemini Google bot»

Еще один аналог – MazAi (Рисунок 1.4).



Рисунок 1.4 – Приложение Telegram-бот «MazAi»

Обладает схожим функционалом, но описание фотографий, которое на выходе получает пользователь, не несёт той конкретики, какая нужна в контексте распознавания достопримечательностей.

Другое приложений из данной категории – «Vision Bot» (Рисунок 1.5).

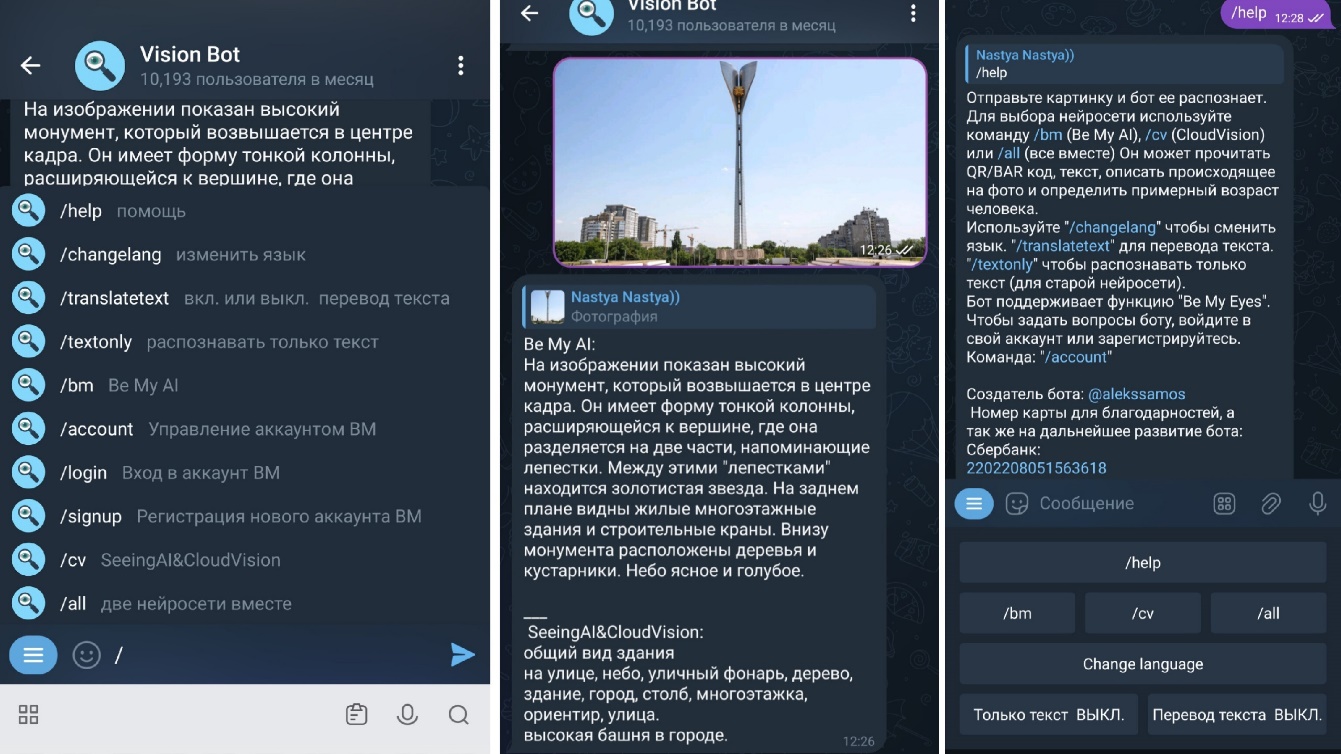


Рисунок 1.5 – Приложение Telegram-бот «Vision Bot»

Данный продукт больше похож на предыдущий тем, что описание изображения поверхностное. Если рассматривать функционал, то у всех трех приложений он примерно одинаковый.

После анализа каждого аналога было решено сравнить характеристики и функционал каждого, а результат представить в виде таблицы (Рисунок 1.6).



Рисунок 1.6 – Таблица сравнения аналогов

### 1.6 Функциональные требования

Итоговый программный продукт должен будет иметь следующие возможности:

1. Распознавание достопримечательности на фотографии и вывод описания пользователю в текстовом виде;
2. Возможность смотреть инструкцию по пользованию ботом – раздел «info»;
3. Приложение должно иметь понятный и удобный интерфейс для обеспечения комфортной работы пользователям.

## 1.7 Обоснование выбора платформы приложения

Было принято решение разрабатывать приложение Telegram-бот. Причиной для выбора послужили удобный и простой интерфейс, а также возможность быстрого обмена информацией между пользователем и приложением. Бота не нужно отдельно устанавливать, достаточно перейти по ссылке и начать работу.

Помимо этого, у ботов нет проблем с обновлениями – если обновить бэкенд бота один раз, то изменения увидят все пользователи. Также ботов можно использовать на любых платформах, где работает Telegram: Android, Windows, Linux и т.д.

### 1.7.1 Выбор языка программирования

В качестве основного языка программирования для разработки был выбран Python. Выбор обоснован тем, что язык предоставляет много библиотек, которые выручат при создании Telegram-бота, а также при работе с нейронными сетями. Ниже приведены некоторые популярные библиотеки глубокого обучения для Python:

* PyTorch — это оптимизированная тензорная библиотека для глубокого обучения с использованием графических процессоров [2];
* Scikit-learn — библиотека с простыми и эффективными инструментами для прогнозного анализа данных [3];
* TensorFlow — библиотека, которая упрощает создание моделей машинного обучения, которые могут работать в любой среде [4];
* Keras — библиотека глубокого обучения, написанная на Python и способная работать поверх TensorFlow [5];
* SciPy — набор математических алгоритмов и удобных функций NumPy. Он добавляет Python значительную мощность, предоставляя пользователю команды и классы высокого уровня для манипулирования и визуализации данных [6].

Также судя по исследованиям, язык имеет высокую популярность и находится на первых местах в рейтинге по версии Google (Рисунок 1.7) [7].

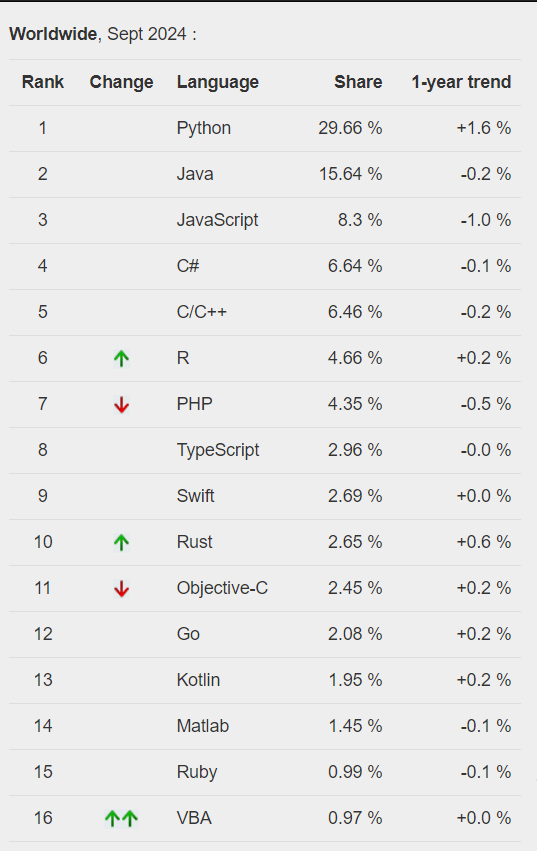


Рисунок 1.7 – PYPL PopularitY of Programming Language

Для создания своего продукта я решила использовать библиотеку PyTorch, так как она предоставляет широкий выбор моделей, методов и модульных элементов для работы с глубоким обучением. Она хорошо подходит для решения задач компьютерного зрения, давая возможность использовать популярные модели. Ниже приведены некоторые из них:

* AlexNet – сверточная нейронная сеть глубиной 8 слоев. Размер входных изображений 224 × 224 пикселей (Рисунок 1.8). Предварительно обученная сеть может классифицировать изображения в 1000 категорий объектов, таких как клавиатура, мышь, карандаш и многие животные. В результате сеть изучила богатые представления функции для широкого спектра изображений [8]. Достаточно простая в использовании архитектура, но в то же время эффективная в решении задач компьютерного зрения. Для успешного обучения требует достаточно большой объем данных.

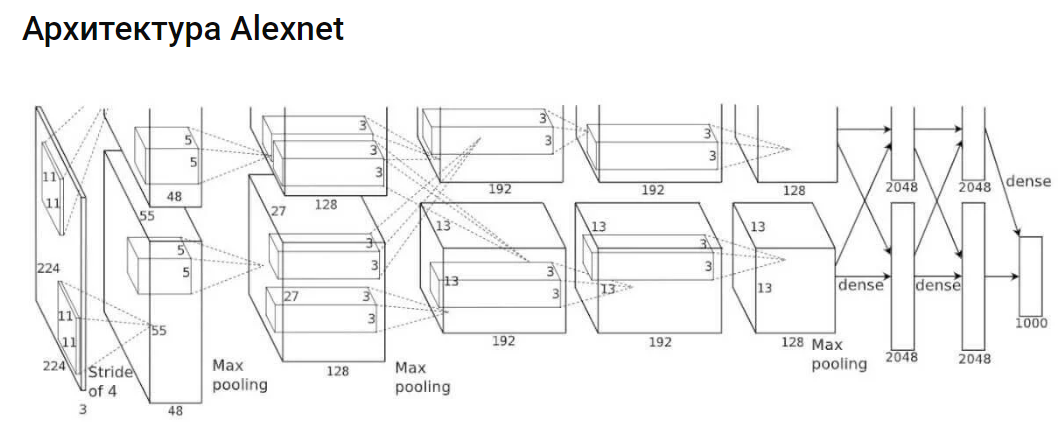


Рисунок 1.8 – Архитектура AlexNet

* ResNet – модель, которая наряду с AlexNet демонстрирует хорошие результаты в распознавании изображений, так как способна извлекать более сложные признаки. Это также является недостатком, потому что ResNet более сложная, чем AlexNet, что делает ее более ресурсоемкой.

Для создания экземпляра ResNet можно использовать различные конструкторы: resnet18, resnet34 resnet50, resnet101 и т.д. Все разработчики моделей внутренне полагаются на базовый класс torchvision.models.resnet.ResNet [9]. Конструкторы различаются количеством слоев, например, resnet101 содержит 99 слоев без учета входных и выходных (Рисунок 1.9).

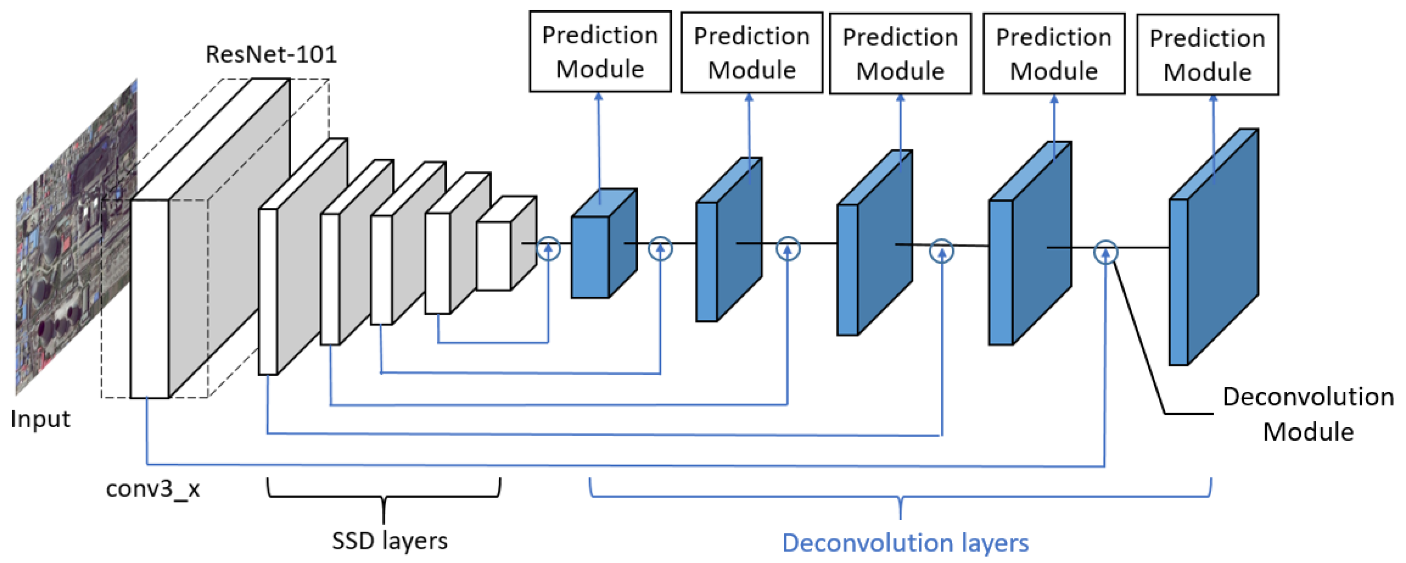


Рисунок 1.9 – Архитектура ResNet-101

* VGG – модель, которая использует сверточные слои с ядрами, идущих один за другим (Рисунок 1.10). Она достаточно проста в использовании, и показывает неплохие результате в решении задач компьютерного зрения.

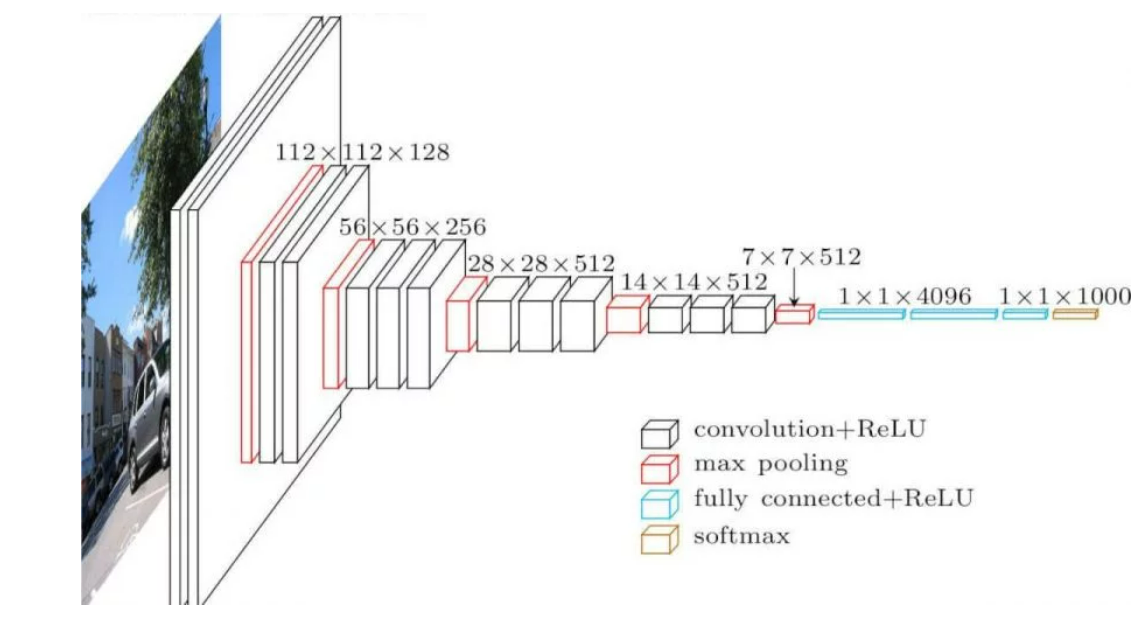


Рисунок 1.10 – Архитектура VGG

* DenseNet – достаточно молодая модель, которая отличается прямым соединением слоев. Она хорошо подходит для задач распознавания изображений, а также для обработки естественного языка. Недостатком данной архитектуры является высокая вычислительная сложность.

Для реализации своего программного продукта я решила остановиться на архитектуре AlexNet благодаря ее простоте и эффективности при работе с изображениями.

### 1.7.2 Выбор среды разработки приложения

Для разработки приложения была выбрана среда разработки Visual Studio Code, так как она характеризуется легкостью использования, но при этом мощностью. Также она предоставляет такие функции, как подсветка синтаксиса, отладка, широкий выбор расширений и т.д. Она может быть установлена на любой платформе: Windows, Linux и macOS. Также она дает возможность работать с практически любым языком программирования.

Еще одно преимущество Visual Studio Code в том, что она поддерживает работу с ноутбуками Jupiter, поэтому начать работу в данной среде достаточно просто.

## 1.8 Выводы и результаты по главе

Целью распознавания изображений является обработка и идентификация визуальных объектов в попытке приблизить их максимально близко к человеческому пониманию.

Целью выпускной работы является создание программного продукта, классифицирующего достопримечательности Ростова-на-Дону. Приложение должно быть реализовано в виде Telegram-бота. Для достижения цели были изучено аналогичное программное обеспечение, был выбран язык программирования, среда разработки, а также библиотека, которая отлично подходит для задач компьютерного зрения.

2. ПРОЕКТИРОВАНИЕ СИСТЕМЫ

2.1 Проектирование системы, UML моделирование

Проектирование программного продукта играет ключевую роль в его разработке, так как позволяет избежать ряда сопутствующих проблем, а также представить продукт в виде схемы еще на начальном этапе. UML моделирование позволит:

* Выявить основные сущности системы, а также определить связи между ними с помощью диаграммы;
* Выявить потенциальные проблемные места в системе и пересмотреть их;
* Оценить способность системы решать поставленные перед ней задачи.

Для визуализации системы используется язык UML – Unified Modeling Language, как следует из названия унифицированный язык моделирования для описания, визуализации и документирования объектно-ориентированных систем в процессе их анализа и проектирования [10].

UML представляет собой графическую нотацию, которая предназначена для моделирования и описания всех процессов, протекающих в процессе разработки. Основу UML представляют диаграммы, которые различаются по типам и предназначены для моделирования различных аспектов разработки [10].

Все диаграммы можно условно разделить на поведенческие и структурные. Поведенческие диаграммы отображают процессы, протекающие в моделируемой среде. Структурные диаграммы отображают элементы, из которых состоит система. При этом одни и те же типы диаграмм могут использоваться как для моделирования бизнес-процессов, так и для непосредственного проектирования архитектуры [10].

Диаграмма – это графическое представление набора элементов, чаще всего изображенного в виде связного графа вершин (сущностей) и путей (связей) [11].

Теоретически диаграмма может включать в себя любую комбинацию сущностей и связей. На практике, однако, используется лишь небольшое число общих комбинаций, состоящих из пяти наиболее часто применяемых представлений архитектуры программных систем. По этой причине UML включает 13 видов диаграмм [11]:

1. Диаграмма классов;

2. Диаграмма объектов;

3. Диаграмма компонентов;

4. Диаграмма составной структуры;

5. Диаграмма вариантов использования;

6. Диаграмма последовательности;

7. Диаграмма коммуникации;

8. Диаграмма состояний;

9. Диаграмма деятельности;

10. Диаграмма размещения;

11. Диаграмма пакетов;

12. Временная диаграмма;

13. Диаграмма обзора взаимодействий.

### 2.1.1 Диаграммы вариантов использования

Для определения пользователей, основного функционала программного продукта и сценариев взаимодействия пользователя с ним будет использована диаграмма вариантов использования.

Диаграмма включает в себя следующие составляющие:

1. Актер (от англ. actor) – использующий систему пользователь. В случае с разрабатываемой системой актеров будет один. На диаграмме актер отображается как подписанный рисунок человека;
2. Вариант использования (прецедент) – элемент диаграммы, являющийся функционалом системы, который доступен пользователю. Прецеденты отражают сценарий использования пользователем приложения;
3. Отношения – связующие между актерами и вариантами использования, а также между вариантами использования друг с другом.

В языке UML на диаграммах вариантов использования поддерживается несколько типов связей между элементами диаграммы [10]:

* Коммуникации (communication);
* Включения (include);
* Расширения (extend);
* Обобщения (generalization).

Связь коммуникации – это связь между вариантом использования и действующим лицом. На языке UML связи коммуникации показывают с помощью однонаправленной ассоциации (сплошной линии со стрелкой). Направление стрелки позволяет понять, кто инициирует коммуникацию [10].

Связь включения применяется в тех ситуациях, когда имеется какой-то либо фрагмент системы, который повторяется более чем в одном варианте использования. С помощью таких связей обычно моделируют многократно используемую функциональность.

Связь расширения применяется при описании изменений в нормальном поведении системы. Она позволяет варианту использования только при необходимости использовать функциональные возможности другого.

К разрабатываемому проекту на этапе проектирования была составлена диаграмма вариантов использования нейронной сети Telegram-ботом (Рисунок 2.1):

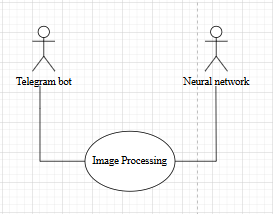


Рисунок 2.1 – Диаграмма вариантов использования нейронной сети Telegram-ботом

К разрабатываемому проекту на этапе проектирования была составлена диаграмма вариантов использования приложения пользователем на основе описанных ранее правил оформления (Рисунок 2.2):

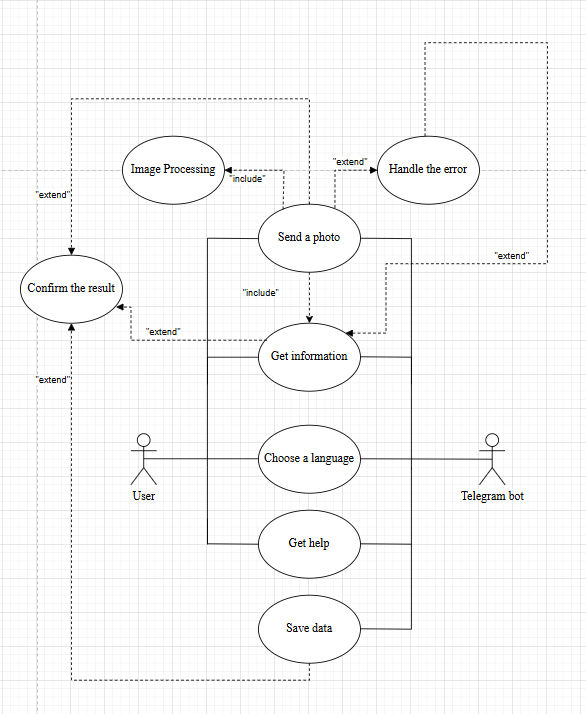


Рисунок 2.2 – Диаграмма вариантов использования приложения пользователем

Таблица 1. Детализация диаграммы вариантов использования приложения пользователем

|  |  |
| --- | --- |
| **Вариант использования** | **Описание** |
| Отправить фотографию | Отправка пользователем фотографии достопримечательности. |
| Обработка изображения | Отправленное пользователем изображение, обработанное с помощью нейронной сети, получает метку класса, то есть название достопримечательности. На основе подобранного названия выбирается описание, которое впоследствии будет возвращено пользователю. |
| Выбрать язык | Пользователю предлагается выбрать язык – русский или английский. На основе этого бот будет возвращать результат в удобном виде. |
| Получить помощь | В разделе «info» описана информация про Telegram-бота, принцип работы и его возможности. |
| Получить информацию | В результате распознавания бот возвращает пользователю класс объекта, а также описание, другие фотографии конкретной достопримечательности, ссылку на карту. Это дает возможность отправителю сориентироваться в городе. |
| Подтвердить результат | После получения результата пользователю предлагается оставить обратную связь и указать, правильно ли было распознано изображение. |
| Обработка ошибок | При возникновении ошибок, пользователю возвращается сообщение об этом. |
| Сохранить данные | После каждого возвращаемого описания информация о правильности/неправильности распознавания фиксируется. |

Чтобы более четко отобразить все варианты использования, была также построена диаграмма вариантов использования, отражающая взаимодействие пользователя и нейронной сети через Telegram-бота (Рисунок 2.3):

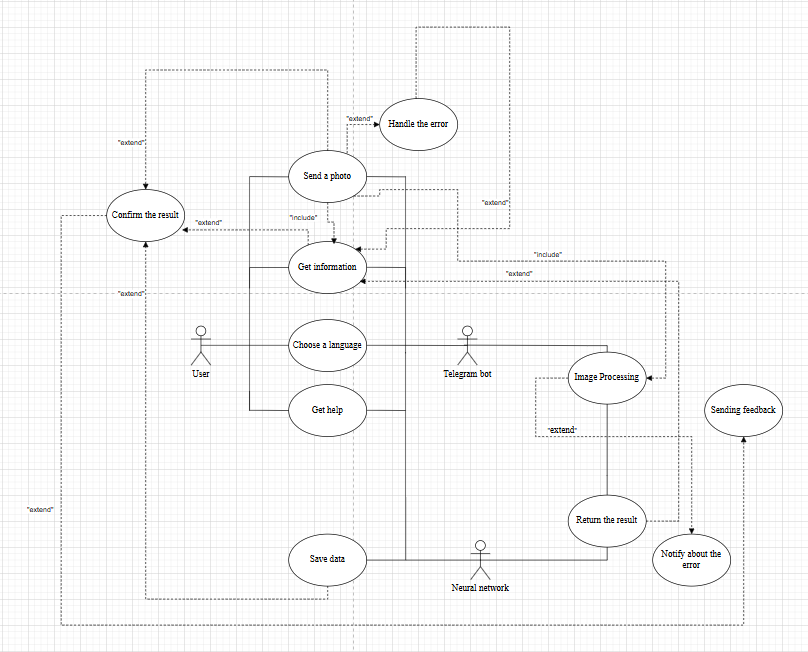


Рисунок 2.3 - Диаграмма вариантов использования нейронной сети пользователем через Telegram-бота

### 2.1.2. Диаграмма классов

Для представления архитектуры программной системы лучше всего представлять в виде диаграммы классов.

Диаграмма классов (class diagram) – диаграмма, предназначенная для представления модели статистической структуры программной системы в терминологии классов объектно-ориентированного программирования. Диаграмма классов является основной логической моделью проектируемой системы [10].

Диаграмма классов определяет типы классов системы и различного рода статические связи, которые существуют между ними. На диаграммах классов изображаются также атрибуты классов, операции классов и ограничения, которые накладываются на связи между классами [10].

Главная цель разработки диаграммы классов – определить сущности и представить их в форме классов с соответствующими атрибутами и операциями [10].

Одна диаграмма классов описывает конкретный аспект системы, а набор диаграмм классов представляет всю систему. По сути, диаграмма классов представляет собой статический вид системы [12].

К разрабатываемому проекту была составлена диаграмма классов (Рисунок 2.4):

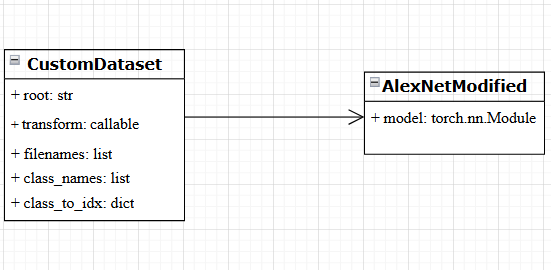


Рисунок 2.4 – Диаграмма классов

Для того, чтобы изобразить большую систему, необходимо учесть множество сущностей. Большое количество компонентов на одной диаграмме может сделать диаграмму нагроможденной, что затрудняет ее чтение.

Вместо моделирования каждой сущности и ее взаимосвязей на одной диаграмме классов лучше использовать несколько диаграмм классов. Разделение системы на несколько диаграмм классов облегчает понимание системы, особенно если каждая диаграмма является графическим представлением определенной части системы [13].

Чтобы более детально описать объекты проектируемой системы и более четко отобразить отношения между ними, была построена расширенная диаграмма классов (Рисунок 2.5):

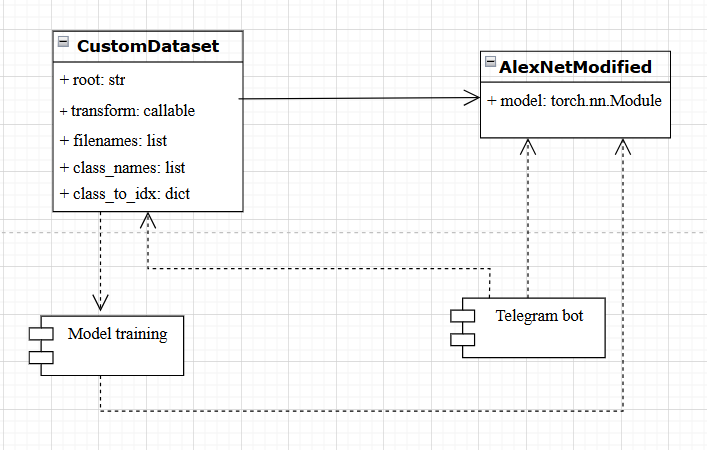


Рисунок 2.5 – Расширенная диаграмма классов

## 2.3 Проектирование интерфейса приложения

Приложение Telegram-бот – это программа, которая помогает автоматизировать различные задачи посредством общения с пользователем через мессенджер Telegram.

Одной из важнейших характеристик Telegram-бота является интерфейс, так как он определяет доверие пользователя к приложению. Красивый дизайн, простое взаимодействие, лаконичные кнопки, а также быстрое выполнение целевых действия напрямую влияют на прирост аудитории, которая использует Telegram-бота.

Интерфейс Telegram-бота может включать следующие элементы:

* Команды – ключевые слова, определяющие функционал бота. Пользователь вправе выбирать нужную команду после ввода символа «/»;
* Клавиатуры – специальные кнопки с репликами или вопросами, которые бот может предложить пользователю на выбор;
* Меню - кнопка, при нажатии на которую пользователь сможет увидеть доступные команды.

Такой набор компонентов помогает обеспечить эффективное и удобное взаимодействие между пользователем и Telegram-ботом.

После определения основных свойств разрабатываемого интерфейса, необходимо создать макет проекта.

Для реализации интерфейса и функционала приложения была выбрана библиотека языка программирования Python telebot. Ниже представлены этапы создания Telegram-бота:

1. Получение токена;
2. Создание имени пользователя;
3. Настройка описания изображения;
4. Настройка меню команд;
5. Реализация начального функционала;
6. Интеграция нейронной сети;
7. Расширение функционала.

После получения специального уникального номера, то есть токена, и создания имени пользователя была реализована страница Telegram-бота, где содержится описания и главное изображение (Рисунок 2.6). Перейдя на главную страницу, пользователь может увидеть имя пользователя, а также вложения – ранее отправленные изображения.



Рисунок 2.6 – Страница бота с основной информацией о нём

Далее было создано меню с основными командами Telegram-бота - /start и /info (Рисунок 2.7).



Рисунок 2.7 – Меню с основными командами Telegram-бота

После команды /start пользователь получает приветственный текст с просьбой выбрать язык. Команда /info позволяет получить краткое руководство по эксплуатации бота и список его возможностей (Рисунок 2.8).

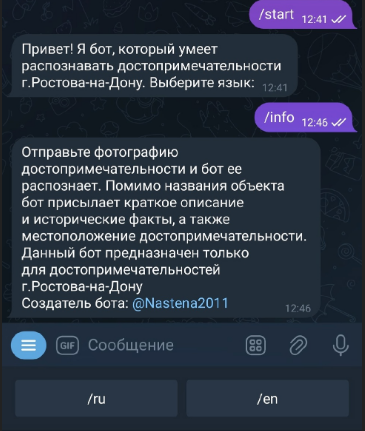


Рисунок 2.8 – Команды «/start», «/info» и кнопки для выбора языка

Затем был реализован ответ Telegram-бота на запрос пользователя. Он представляет собой текстовое описание конкретной достопримечательности, а также получение обратной связи от пользователя (Рисунок 2.9).

Получение обратной связи о том, был ли правильно распознан объект, является важной частью работы проектируемой системы. Это позволяет не только контролировать работу приложения, но и улучшать ее в будущем. Хранение и получение неправильно распознанных объектов позволяет выявить ошибки в работе модели и скорректировать ее. Также учет мнения пользователей повышает уровень их доверия к продукту, что также является сильным преимуществом любого программного сервиса.

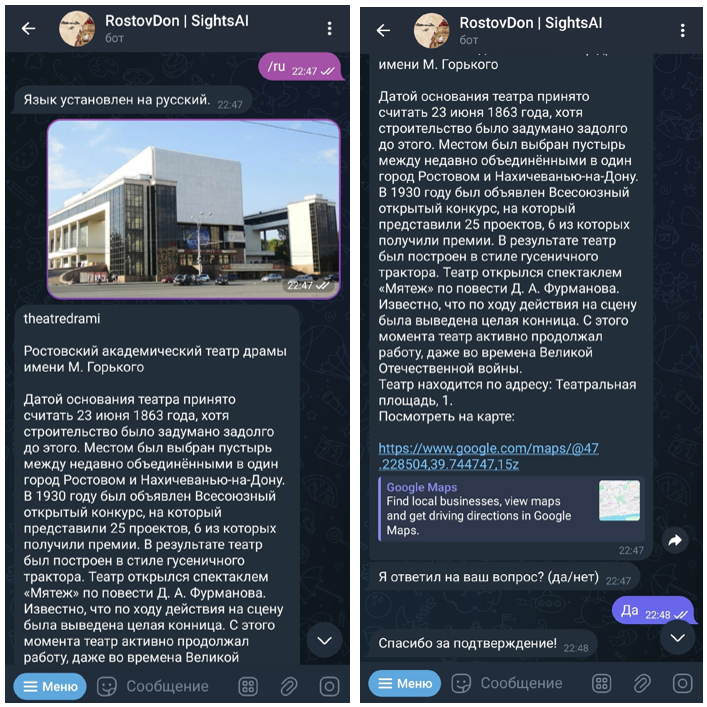


Рисунок 2.9 – Ответ Telegram-бота на фотографию пользователя

2.4 Выводы и результаты по главе

Таким образом были построены основные UML диаграммы, которые описывают основные аспекты разрабатываемого приложения. Также был спроектирован интерфейс Telegram-бота.

# 3. РЕАЛИЗАЦИЯ СИСТЕМЫ

## 3.1 Реализация нейронной сети

Задача классификации изображений заключается в том, чтобы получить изображение и вывести его класс или группы наиболее вероятных классов. Для человека распознавание объектов является неотъемлемой частью повседневной жизни, в то время как такая возможность недоступна электронным устройствам.

Для того, чтобы сделать распознавание изображений возможным для вычислительной техники, нужно воспользоваться концепцией нейронных сетей. Работа с фотографиями – это одна из основных задач сверточных нейронных сетей.

Стоит начать с объяснения того, как компьютер «видит» цветное изображение. Каждый пиксел цветного цифрового изображения состоит из трех цветов – красного, зеленого и синего [14]. Каждый цвет представляет собой матрицу значений от 0 до 255.

Эффективное решение упомянутой задачи происходит с помощью процесса, называемого сверткой. Свертка – это математический аналог анализа изображения с помощью ряда увеличительных стекол, каждое из которых предназначено для разных целей [14]. Перемещение стекол осуществляется с помощью математических законов, что позволяет заметить такие паттерны, как линии, текстуры, закругленные края и т.д. Вычисления происходят на основе значений пикселов, например, 0 вокруг больших значений означает край.

После обнаружения локальных паттернов с помощью свертки нейроны скрытого слоя начинают соединять важные края и отфильтровывать информацию, не имеющую отношения к целевому выходу [14]. Слой нейронной сети – это множество нейронных элементов, на которые в каждый такт времени параллельно поступает информация от других нейронных элементов сети [15]. Средний слой — «первый» — называется скрытым, так как только на этом слое значения явно не отображаются во время обучения [14].

AlexNet – глубокая сверточная нейронная сеть, распознающая изображения. В свое время она достигла невероятных результатов на наборе данных ImageNet.

В 2012 году именно эта конфигурация нейронной сети и осуществила переворот в распознавании изображений, так как была видоизменена структура формирования сверточных слоев [17].

AlexNet прошел обучение работе с более чем миллионом изображений и может классифицировать изображения по 1000 категориям объектов (таким как клавиатура, кофейная кружка, карандаш и многие животные). Сеть изучила богатые представления функций для широкого спектра изображений. Сеть принимает изображение в качестве входных данных и выводит метку объекта на изображении вместе с вероятностями для каждой из категорий объектов [16].

Самым первым элементом сети является непосредственно изображение (размерами 224\*224 пикселя). Значение «3» характеризует факт определения цветов в изображении палитрой RGB (размерность вектора каждого пикселя) [17].

Размерность окрестности свертки составляет 11\*11. На втором блоке появляется число 96 Оно определяет количество различных сверток, которые вычисляются в каждом пикселе изображения. Далее, слой за слоем, благодаря пулингу, уменьшается размерность изображения. В конечном счете, изображение становится стянутым до размеров 1\*1 пиксель [17].

Таким образом, постоянно уменьшая размерность самого изображения, была получена его векторизованная форма. Иными словами, в автоматическом режиме были сгенерированы признаки изображения (так как не теряются взаимосвязи между соседними пикселями, их расположением, цветом и так далее) [17].

После векторизации изображения, вектор можно использовать в любых доступных исследователю моделях. Например, добавив всего 2 – 3 полносвязных нейронных слоя (которые уже решают задачу классификации), можно получить нейронную сеть для распознавания объектов на изображении [17].

Результат работы каждого слоя нейронной сети можно условно интерпретировать (Рисунок 3.1) [17].

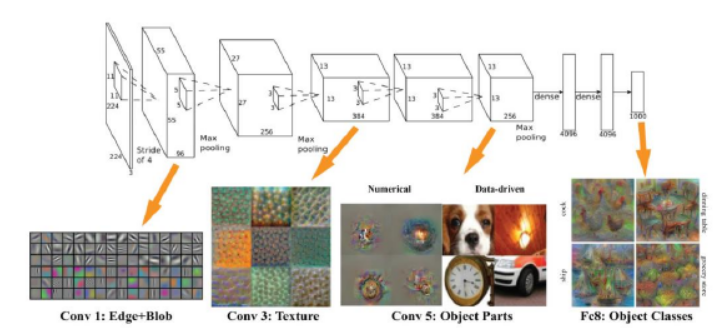


Рисунок 3.1 – Интерпретация результатов работы нейронной сети

Реализация нейронной сети включает в себя следующие этапы:

1. Сбор датасета;
2. Формирование логики нейронной сети
3. Реализация нейронной сети.

### Создание набора данных

Для успешной реализации нейронной сети необходимо было получить некоторое количество фотографий достопримечательностей и объединить их в набор данных для обучения. Сбор датасета происходил по следующему алгоритму:

1. Определиться с изображениями, которые будет классифицировать нейронная сеть;
2. Сделать фотографии выбранных объектов;
3. Выбрать лучшие для каждого класса;
4. Создать директорию и определить внутри еще две – train и val. Затем перенести изображения в эти директории в соотношении 80% на 20%.

На начальном этапе было необходимо получить набор оригинальных фотографий достопримечательностей. Для этого были сняты вручную изображения объектов при различных ракурсах и освещениях (Рисунок 3.2). Сбор этих данных осуществлялся в ходе посещения достопримечательностей.



Рисунок 3.2 – Примеры фотографий Администрации г.Ростова-на-Дону

Далее проводилась ручная фильтрация изображений – удаление неудачных или размытых кадров. В результате получилось 619 изображений.

Необходимо обеспечить, чтобы нейронная сеть могла классифицировать изображения с разных ракурсов. Поэтому количество изображений для каждого класса должно быть достаточным для того, чтобы обеспечить высокую вероятность правильного соотнесения объекта на фото к нужному классу. Необходимо учесть индивидуальные особенности достопримечательности, такие как мелкие детали, края и прочие особенности. Также от размера датасета зависит обобщающая способность модели.

Обобщаемость относится к разнице в производительности модели при оценке по ранее просмотренным данным (обучающим данным) по сравнению с данными, которые она никогда не видела ранее. Модели с плохой обобщаемостью показывают значительно большую точность на выборке для обучения, чем на тестовой выборке. В этом заключается эффект переобучения. Поэтому, как правило, в каждую эпоху обучения необходимо соотносить точность (или эффективность) на выборке для обучения с точностью (или эффективностью) на проверочной выборке. Для построения полезных моделей глубокого обучения ошибка на проверочных данных должна продолжать уменьшаться вместе с ошибкой на обучающей выборке. Увеличение данных – очень мощный метод достижения этого [18].

Аугментация данных широко применяется при обучении современных нейронных сетей, в том числе предназначенных для решения задач компьютерного зрения [19, 20]. Она позволяет увеличить объём обучающей выборки и тем самым снижает риск переобучения модели, не требуя при этом дополнительного сбора новых обучающих примеров [21].

Увеличение набора данных обеспечивается трансформацией уже существующих изображений с помощью следующих методов (Листинг 3.1):

**Листинг 3.1 - Аугментация данных**

data\_transforms = {

'train': transforms.Compose ([

transforms.RandomResizedCrop (224),

transforms.RandomHorizontalFlip (),

transforms.ColorJitter (brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),

# Случайное изменение масштаба

transforms.RandomAffine (degrees=0, translate=(0.1, 0.1), scale=(0.9, 1.1)),

transforms.RandomRotation (degrees= (-15, 15)),

transforms.ToTensor (),

transforms.Normalize ([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

]),

'val': transforms.Compose([

transforms.Resize(256),

transforms.CenterCrop (224),

transforms.ToTensor(),

transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

]),

}

Аугментация проводилась с помощью библиотеки torchvision, которая предоставляет широкий набор преобразований и полезных инструментов. Каждое примененное преобразование выполняет определенную функцию:

1. RandomResizedCrop – отвечает за изменение масштаба и ракурса съемки;
2. RandomHorizontalFlip - добавляет отражение по горизонтали;
3. ColorJitter - случайным образом меняет яркость, контрастность, насыщенность и оттенок изображения [30];
4. RandomAffine – добавляет геометрические искажения;
5. RandomRotation - поворот на некоторый угол [29].

Посмотреть на изображения, получившиеся в результате аугментации данных можно отобразив их с помощью библиотеки matplotlib (Рисунок 3.3). Для этого был написан метод imshow() (Листинг 3.2).

**Листинг 3.2 – Отображение получившихся изображений с соответствующими метками**

def imshow(images, labels, class\_names):

"""Отображает изображения из батча"""

images = images.numpy().transpose((0, 2, 3, 1)) # Изменяем размерность для отображения

plt.figure(figsize=(12, 6))

num\_images = min(len(images), 8)

for i in range(num\_images):

ax = plt.subplot(2, 4, i + 1)

plt.imshow(images[i].clip(0, 1)) plt.title(class\_names[labels[i]] if labels[i] is not None else "Нет метки")

plt.axis('off')

plt.show()



Рисунок 3.3 – Отображения получившихся изображений с соответствующими метками

Также возможно посмотреть на изображение до и после применения преобразований (Рисунок 3.4). Для этого был написан метод show\_augmentation\_example () (Листинг 3.3).

**Листинг 3.3 – Отображение** **изображения до и после аугментации**

def show\_augmentation\_example(dataset, idx=0):

original\_img = Image.open(os.path.join(dataset.root, dataset.filenames[idx])).convert('RGB')

# Получаем аугментированное изображение

augmented\_img = dataset.transform(original\_img) if dataset.transform else original\_img

# Преобразуем тензор обратно в изображение для отображения

if isinstance(augmented\_img, torch.Tensor):

augmented\_img = augmented\_img.numpy().transpose((1, 2, 0))

augmented\_img = numpy.clip(augmented\_img, 0, 1)

# Визуализация

plt.figure(figsize= (10, 5))

plt.subplot(1, 2, 1)

plt.imshow(original\_img)

plt.title('Оригинальное изображение')

plt.axis('off')

plt.subplot(1, 2, 2)

plt.imshow(augmented\_img)

plt.title('Изображение после аугментации')

plt.axis('off')

plt.show()



Рисунок 3.4 – Отображение изображения до и после аугментации

Если сравнить исходные изображения с новыми, то видно, что аугментация позволяет создать разнообразные изображения, сохраняя их содержание.

В результате получился набор данных для обучения, а также набор изображений, не участвующих в обучении, которые распределены в соотношении 80% на 20%. Изображения были размечены в соответствии с классами. В общем случае получилось 1857 изображений 32 объектов.

### 3.1.2 Формирование логики нейронной сети

Для построения модели нейронной сети используется библиотека PyTorch, которая представляет все необходимые возможности для работы со сверточными сетями.

Если задачу реализации нейронной сети разделить на несколько небольших, то получаем следующий алгоритм:

1. Построение модели нейронной сети;
2. Реализация и настройка обучения модели;
3. Анализ работы модели на проверочных данных;
4. Сохранение обученной модели.

Построение модели нейронной сети производилось на основе архитектуры AlexNet. Модель получает на вход количество классов, то есть количество объектов, которые необходимо классифицировать.

В AlexNet содержится классификатор, который имеет 8 слоев. Последний слой отображает признаки 1000 классов набора данных ImageNet. Именно к этому слою идет обращение, а на его месте создается новый линейный слой с указанием количества классов согласно решаемой задачи. Данный метод позволяет адаптировать AlexNet к решаемой задачи распознавания достопримечательностей (Листинг 3.4).

**Листинг 3.4 – Класс модели нейронной сети**

class AlexNetModified(nn.Module):

def \_\_init\_\_(self, num\_classes):

super (AlexNetModified, self).\_\_init\_\_()

self.model = models.alexnet(weights=AlexNet\_Weights.DEFAULT)

self.model.classifier [6] = nn.Linear(self.model.classifier[6].in\_features, num\_classes)

def forward(self, x):

return self.model(x)

Далее следует этап реализации и настройки обучения модели. Для этого необходимо определить следующие параметры:

* функция потерь (loss) или целевая функция оценивает отклонение результатов обучения нейронной сети от ожидаемых – это так называемые потери. Целевая функция позволяет корректировать обучение в направлении минимизации потерь. От того, что будет выбрано в качестве целевой функции зависит и результат обучения нейронной сети [22]. Ниже приведены примеры функций потерь, реализованных в PyTorch:

1. nn.CrossEntropyLoss: Cross Entropy Loss (перекрестная энтропия) – используется для многоклассовой классификации. Вычисляет потери на основе логарифма вероятностей предсказанных классов и целевых классов [24];
2. nn.MSELoss: Mean Squared Error Loss (средняя квадратичная ошибка) – вычисляет среднеквадратичную разность между предсказанными и целевыми значениями. Обычно используется в задачах регрессии [24];
3. nn.NLLLoss: Negative Log Likelihood Loss отрицательное логарифмическое правдоподобие) – также используется для задач многоклассовой классификации. Ожидает на вход логарифмы вероятностей предсказанных классов и целевых классов [24];
4. nn.BCELoss: Binary Cross Entropy Loss (бинарная перекрестная энтропия) – применяется в задачах бинарной классификации. Вычисляет потери на основе логарифма вероятности предсказания истинного класса [24];
5. nn.BCEWithLogitsLoss: это альтернатива nn.BCELoss, которая объединяет в себе функцию активации Sigmoid и функцию потерь BCELoss. Обычно используется для бинарной классификации, когда в выходном слое модели не применяется функция активации [24];
6. nn.L1Loss: Mean Absolute Error Loss (средняя абсолютная ошибка) – вычисляет среднюю абсолютную разность между предсказанными и целевыми значениями. Также применяется в задачах регрессии;
7. nn.SmoothL1Loss: так же известна как Huber Loss или L1 Loss с использованием окна сглаживания. Сочетает в себе свойства L1 и L2 Loss и обеспечивает плавный переход между ними.

Для решения задачи в качестве функции потерь была выбрана CrossEntropyLoss(), которая позволяет оценить правдоподобие предсказаний в задаче многоклассовой классификации.

* оптимизатор (optimizer) реализует основной алгоритм глубокого обучения - алгоритм обратного распространения ошибки - механизм, с помощью которого нейронная сеть обновляет так называемые веса - параметры обучения – на основе данных, получаемых от функции потерь и настройки скорости обучения. Существуют несколько видов оптимизаторов, которые реализуют разные алгоритмы оптимизации на основе вычисления градиента, чтобы достичь наилучшего результата по нахождению минимума функции потерь [22]:

1. torch.optim.SGD: Stochastic Gradient Descent (стохастический градиентный спуск) – это один из наиболее распространенных оптимизаторов. Обновляет параметры модели, вычисляя градиенты по каждому пакету обучающих данных. Позволяет настроить скорость обучения и момент [24];
2. torch.optim.Adam: Adam (адаптивный метод оптимизации с моментами) – эффективный оптимизатор, сочетающий в себе идеи стохастического градиентного спуска и адаптивного градиентного спуска. Автоматически адаптирует скорость обучения для каждого параметра на основе истории градиентов [24];
3. torch.optim.RMSprop: RMSprop (адаптивный метод оптимизации с квадратичным средним градиентов) – адаптивный оптимизатор, который также адаптирует скорость обучения для каждого параметра, учитывая историю квадратичных средних градиентов [24];
4. torch.optim.Adagrad: Adagrad (адаптивный градиентный метод) – оптимизатор, который адаптирует скорость обучения для каждого параметра, учитывая историю суммы квадратов градиентов. Предназначен для эффективной работы с разреженными градиентами [24];
5. torch.optim.Adadelta: Adadelta – оптимизатор, основанный на методе AdaGrad, который дополнительно решает проблему уменьшения скорости обучения со временем. Использует историю средних квадратов изменений параметров для адаптации скорости обучения [24];
6. torch.optim.AdamW: AdamW – вариант оптимизатора Adam, который добавляет технику сокращения весов (weight decay) в процесс оптимизации. Сокращение весов помогает предотвратить переобучение модели [24];
7. torch.optim.SparseAdam: SparseAdam – оптимизатор, разработанный специально для работы с разреженными градиентами. Предоставляет более эффективный способ работы с разреженными данными, сокращая вычислительные затраты [24].

Для контроля процесса обучения в рамках конкретной задачи был выбран оптимизатор optim.Adam().

* метрика (metrics) служит для мониторинга качества предсказаний на этапе обучения и тестирования. Для анализа данных могут использоваться метрики, такие как precision, recall, accuracy. Метрикой, использовавшейся в процессе реализации нейронной стала accuracy (точность). Она представляет из себя долю правильно предсказанных объектов от общего числа объектов в выборке. Математически это можно представить следующим образом:

где true positive (TP) — положительный класс распознан как положительный [23];

false positive (FP) — ошибка 1-го вида, когда отрицательный класс распознается как положительный [23];

true negative (TN) — отрицательный класс распознается как отрицательный [23];

false negative (FN) — ошибка 2-го вида, когда положительный класс распознается как отрицательный [23].

Подача данных в нейронную сеть производится небольшими порциями с определенным количеством изображений. Такая порция называется мини-пакетом или батчем. Размер такого пакета определяем при помощи параметра batch\_size. Этот гиперпараметр требует настройки, так как влияет на скорость обучения.

Процесс тренировки нейронной сети также регулировался следующими функциями:

* EarlyStopping - чтобы остановить процесс обучения, если точность обучения на проверочной выборке не будет улучшаться в течение n эпох обучения [22];
* ReduceLROnPlateau - чтобы регулировать скорость обучения, если точность обучения на проверочной выборке не будет улучшаться в течение n эпох обучения [22].

В ходе обучения видно, как меняются значения обучающей и проверочной 'loss', а также 'accuracy' (Рисунок 3.5).

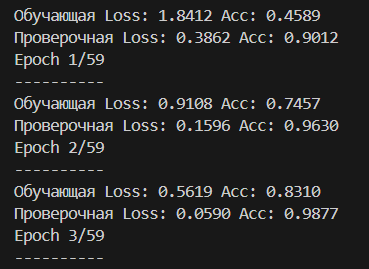


Рисунок 3.5 – Процесс обучения нейронной сети

Для анализа значений функции потерь был построен график ее изменения (Рисунок 3.6). На графике представлены кривая для обучающей выборки и для валидационной. Отображение осуществляется с помощью метода loss\_function\_graph () (Листинг 3.5).

**Листинг 3.5 – Построение графика функции потерь**

# Функция для построения графика функции потерь

def loss\_function\_graph(train\_loss\_history, val\_loss\_history):

plt.figure(figsize= (10, 5))

plt.plot(train\_loss\_history, label = 'Функция потерь на обучающем наборе')

plt.plot(val\_loss\_history, label = 'Функция потерь на проверочном наборе')

plt.title('График функции потерь')

plt.xlabel('Эпохи')

plt.ylabel('Потеря')

plt.legend()

plt.grid(True)

plt.show()

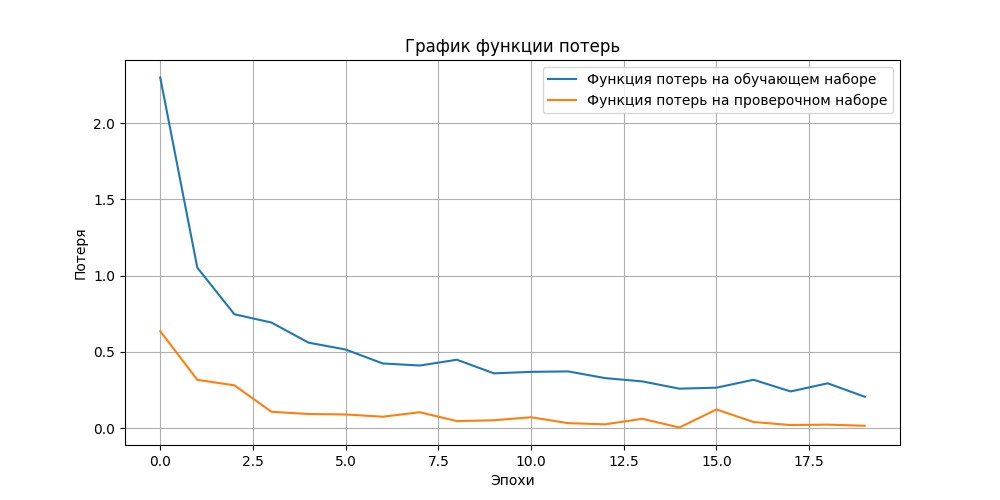


Рисунок 3.6 – График изменения функции потерь

Также была построена матрица ошибок для анализа результата работы модели (Рисунок 3.7). Она позволяет оценить распределение правильных и ошибочных прогнозов по каждому классу. Диагональ матрицы показывает количество правильно распознанных объектов, а остальные – случаи ошибок. Построение матрицы производилось с помощью метода plot\_confusion\_matrix(), представленном в Приложении А.

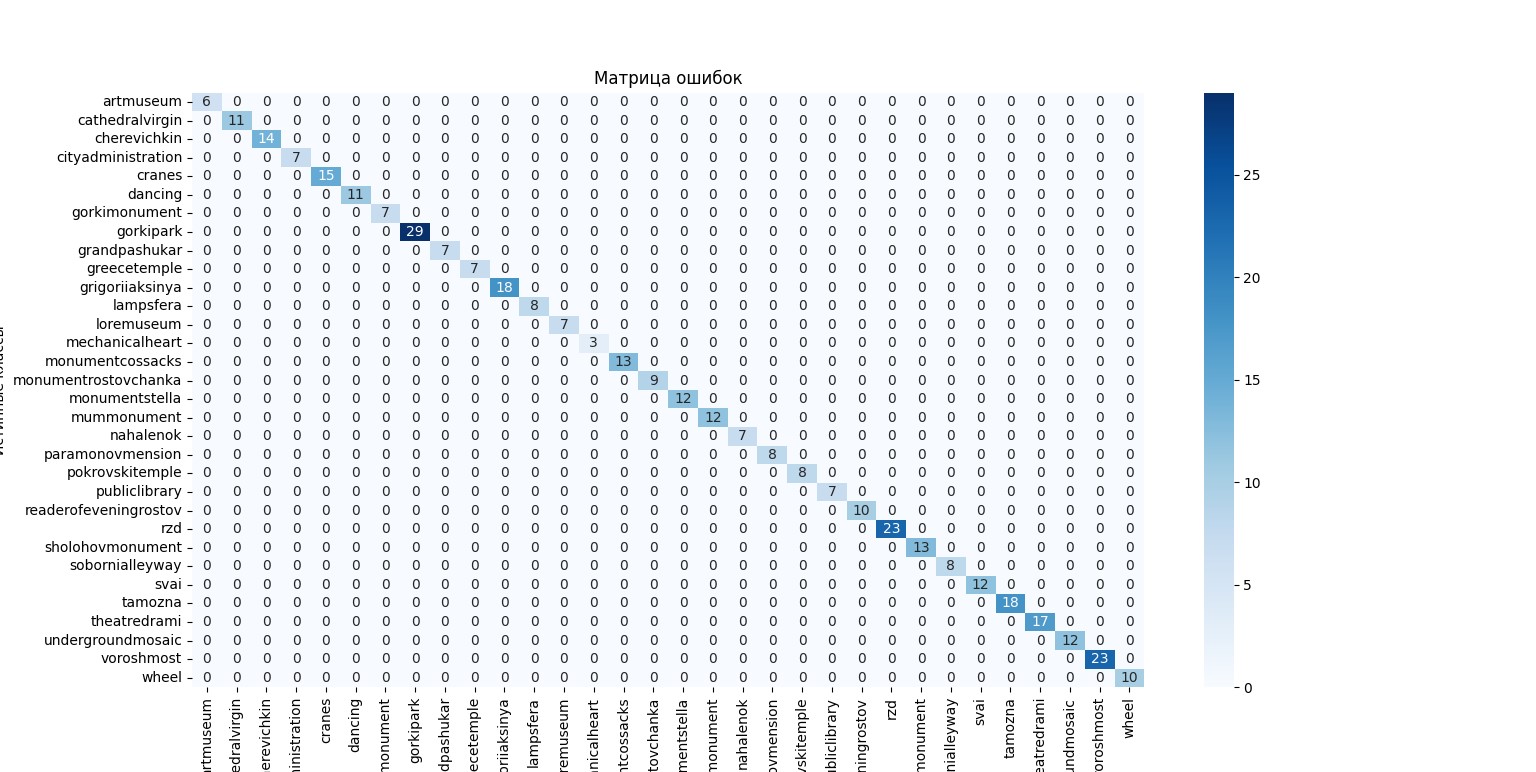


Рисунок 3.7 – Матрица ошибок нейронной сети

Нейронная сеть обучалась на тренировочной выборке, что составляет 80% от общего числа изображений, остальная часть набора данных – изображения, которые модель ранее не видела. Разделение данных является стандартным подходом в машинном обучении. Такие данные нужны для проверки производительности модели, так как позволяют оценивать ее обобщающую способность, тем самым предотвратить переобучение.

## 3.2 Реализация Telegram-бота

Разработка Telegram-бота проходила в несколько этапов:

1. Создание меню выбора языка;
2. Реализация команд /start и /info;
3. Отправка и получение фотографии;
4. Реализация возможности получения обратной связи от пользователя;
5. Обработка ошибок.

Разработка приложения Telegram-бота для распознавания достопримечательностей была обусловлена тем, чтобы предоставить удобный инструмент как для жителей города Ростова-на-Дону, так и для его гостей. Данный продукт будет полезен для иностранных студентов города, которые интересуются культурой и историей города. Это стало поводом для создания меню выбора языка.

Меню выбора языка предоставляет возможность посетителю выбрать из двух возможных языков – русский или английский. Пользователю эта функция предоставляется в виде клавиатуры из двух кнопок. Такой подход повышает уровень доверия к приложению, так как быстрота и комфортное использование является главным достоинством подобных приложений.

Реализация клавиатуры выбора языка происходила путем создания объекта ReplyKeyboardMarkup, который и представляется клавиатурой. Она состоит из кнопок, и может подстраиваться под размеры экрана устройства (Листинг 3.6)

**Листинг 3.6 – Создание клавиатуры выбора языка**

# Функция для создания клавиатуры выбора языка

def language\_keyboard():

keyboard = telebot.types.ReplyKeyboardMarkup(resize\_keyboard=True)

button\_ru = telebot.types.KeyboardButton('/ru')

button\_en = telebot.types.KeyboardButton('/en')

keyboard.add(button\_ru, button\_en)

return keyboard

Далее были реализованы команды две команды из основного меню Telegram-бота - /start и /info. При отправке первой команды, бот вернет приветственное сообщение. Написание программного кода к ней происходило с помощью декоратора библиотеки telebot @bot.message\_handler(commands=['start']). Это означает, что функция вызывается тогда, когда пользователь отправит боту команду /start. После этого в чате появится сообщение «Привет! Я бот, который умеет распознавать достопримечательности г. Ростова-на-Дону.», а также предложение выбрать язык (Листинг 3.7).

**Листинг 3.7 – Реализация команды /start**

@bot.message\_handler(commands=['start'])

def start\_message(message):

bot.send\_message(message.chat.id, 'Привет! Я бот, который умеет распознавать достопримечательности г.Ростова-на-Дону. '

'Выберите язык:', reply\_markup=language\_keyboard())

Аналогично был реализован программный код для команды /info (Листинг 3.8).

**Листинг 3.8 – Реализация команды /info**

@bot.message\_handler(commands=['info'])

def help\_message(message):

# Раздел "помощь"

language = user\_languages.get(message.chat.id, 'ru') # По умолчанию русский

if language == 'ru':

bot.send\_message(message.chat.id, 'Отправьте фотографию достопримечательности и бот ее распознает. Помимо названия объекта бот присылает краткое описание и исторические факты, а также местоположение достопримечательности. Данный бот предназначен только для достопримечательностей г.Ростова-на-Дону\nСоздатель бота: @Nastena2011')

else:

bot.send\_message(message.chat.id, 'Send a photo of a landmark and the bot will recognize it. In addition to the name of the object, the bot sends a short description and historical facts, as well as the location of the landmark. This bot is intended only for landmarks in Rostov-on-Don\nBot creator: @Nastena2011')

Для того, чтобы обеспечить отправку фотографии Telegram-боту и ее обработку с возвратом результата распознавания, были написаны методы get\_photo() и repeat\_all\_messages().

Метод get\_photo() отвечает за получение содержимого фотографии, отправленной пользователем (Листинг 3.9).

**Листинг 3.9 – Метод get\_photo()**

def get\_photo(message):

photo = message.photo[1].file\_id

file\_info = bot.get\_file(photo)

file\_content = bot.download\_file(file\_info.file\_path)

return file\_content

Метод repeat\_all\_messages() задает логику обработки фотографии и выдачи результата пользователю, соответственно определяет основной функционал Telegram-бота. На вход методу подается содержимое сообщения, которое отправил пользователь. Отправленное изображение подготавливается для подачи в нейронную сеть, путем применения преобразований (Листинг 3.10).

**Листинг 3.10 – Преобразования к изображению для подачи в нейронную сеть**

transform = transforms.Compose(

[transforms.Resize(224),

transforms.ToTensor(),

transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])])

Затем происходит определение класса объекта нейронной сетью, и бот получает предсказания. В результате пользователю возвращается имя класса, и описание объекта. Также может быть отправлено сообщение об ошибке, если объект не относится ни к одному из классов. Вместе с описанием объекта Telegram-бот отправляет похожие фотографии в виде группы, а также ссылку на геопозицию объекта. Последнее сообщение бота содержит вопрос: «Я ответил на ваш вопрос? (да/нет)». Это нужно для получения обратной связи о неправильно распознанных достопримечательностях. Код, реализующий данный фрагмент представлен в Приложении Б.

Получение обратной связи от пользователя является очень полезной частью функционала, так как позволяет систематически улучшать качество работы Telegram-бота, а именно повышение точности распознавания достопримечательностей на фотографии.

Чтобы оставить обратную связь, клиент может ввести название объекта или ответить «Я не знаю». Данные сохраняются в файл, и могут быть в дальнейшем использованы для повышения надежности приложения. Реализация данного функционала осуществляется методом process\_incorrect\_data(), представленном в Приложении В.

3.3 Выводы и результаты по главе

Была реализована нейронная сеть и ее процесс обучения с использованием библиотеки PyTorch. Все необходимые задачи были выполнены, а именно:

1. Был собран и подготовлен набор данных с фотографиями достопримечательностей;
2. Реализована модель нейронной сети, подходящей под конкретную задачу, на основе архитектуры AlexNet;
3. Настроен процесс обучения через функцию потерь, оптимизатор, метрики, функций обратного вызова и гиперпараметров;
4. Модель показала хорошие результате на данных, которых не видела ранее.

Также был реализован функционал Telegram-бота такой как:

1. Возврат пользователю названия достопримечательности;
2. Возврат описания объекта с историческими фактами о нем;
3. Отправка ссылки на геопозицию объекта;
4. Отправка похожих фотографий распознанного объекта;
5. Получение обратной связи от пользователя;
6. Сохранение неправильно распознанных объектов для дальнейшего улучшения модели.

# БЕЗОПАСНОСТЬ ЖИЗНЕДЕЯТЕЛЬНОСТИ

## 4. 1. Основные требования и определения

Пользователь компьютера – работник, использующий компьютер для решения производственных задач в рамках своей должностной инструкции [31].

Рабочее место пользователя персонального компьютера – место, где пользователь должен находиться в связи с выполнением поставленных задач, предусматривающих работу за компьютером.

Внедрение ЭВМ имеет как положительные, так и отрицательные моменты. С одной стороны, это обеспечение более высокой эффективности производства за счет совершенствования технологического процесса и повышение производительности труда, а с другой - увеличение нагрузки на работающих в связи с интенсификацией производственной деятельности и специфическими условиями труда [31]. Это может стать причиной появления негативных последствий, которые отразятся на здоровье.

Негативные последствия при работе с компьютером выражаются в следующем: снижение остроты зрения; замедленная перефокусировка; «раздвоение» предметов; быстрое утомление при чтении; жжение и давление в глазах; боли в области глазниц и лба; покраснение глаз. Перечисленные явления обычно объединяют термином "астенопия" (в буквальном переводе – отсутствие силы зрения) [33].

Таким образом, если пользователю выработать правильную культуру взаимодействия, то негативные эффекты можно существенно уменьшить, иногда исключить [32].

* 1. Безопасность на рабочем месте

Перед началом определения методов и начала разработки необходимо ознакомиться с правилами безопасности при работе с персональным компьютером. Для этого следует проверить место работы:

* Освещение при работе за компьютером – приглушенный рассеянный свет.
* Экран монитора должен быть чистым и без повреждений.
* Если работа происходит в очках, то они тоже должны быть абсолютно чистыми.
* Обязательно исключить наличие оголенных проводов и кабелей.
* Убедиться, что на системном блоке отсутствуют лишние предметы и охлаждение не заблокировано.
* Проверить наличие влаги и, если она есть, дождаться ее высыхания перед запуском техники.

После проверки рабочего места необходимо соблюдать следующие правила:

* Нельзя размещать еду или воду рядом с рабочим местом.
* Во время работы не чистить компьютер и не снимать корпус системного блока.
* Избегать контакта с токопроводящими предметами.
* В случае запаха гари немедленно выключить устройство и обратиться к специалистам.
* Сидеть прямо, с упором спины в спинку стула, и локти согнуты под прямым углом.
* Обеспечить постоянное освещение в помещении.
* Создать комфортное рабочее места с применением специальной мебели для длительной статической работы.
* Разместить монитор компьютера на расстоянии не менее метра от глаз и правильно расположить глаза пользователя на верхней трети монитора, для создания прямого угла при удобной сидячей позе.
* Необходимо проводить перерывы в работе каждые 40–60 минут с длительностью от 5 до 10 минут для разминки кистей рук, шеи, спины и выполнять гимнастику глаз для профилактики различных заболеваний.

5. ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ

Целью выпускной квалификационной работы является создание приложение Telegram-бота для распознавания достопримечательностей города Ростова-на-Дону. Созданное приложение позволит пользователям распознать достопримечательность по фотографии и получить ее краткое описание. Использование данного стека технологий бесплатно для разработчика и не требует вложений.

## 6. 1 Определение стоимости разработки

Оценка стоимости разработки приложения начинается с подробного планирования работ, выполняемых на каждом этапе проектирования. Поскольку в процессе создания использовались исключительно бесплатные инструменты, основным ресурсом, влияющим на затраты, являлось время, потраченное на реализацию. Путем определения временных промежутков, затраченных на каждый из этапов разработки, можно вычислить общее время, вложенное в создание конечного продукта. Результаты этих расчетов представлены в Таблице 6.1.

Таблица 6.1. Временные промежутки каждого этапа работы

|  |  |
| --- | --- |
| **Этап** | **Затраченное время (в днях)** |
| Анализ предметной области и оценка систем аналогов | 11 |
| Формирование функциональных требований к системе | 15 |
| Определение средств разработки приложения | 12 |
| Проектирование визуальной части | 7 |
| Проектирование системы, создание UML диаграмм | 25 |
| Создание набора данных | 28 |
| Реализация нейронной сети | 30 |
| Реализация Telegram-бота | 20 |
| Тестирование приложения | 15 |
| Итого: | 163 |

Исходя из полученных расчетов можно посчитать затраты на разработку Telegram-бота. Путем анализа рынка труда, взята заработная плата начинающего IT-специалиста на полставки, графиком 5/2, длительностью рабочего дня в 4 часа и заработной платой 35000 рублей в месяц. Из этих данных можно рассчитать стоимость одного дня работы.

Для этого разделим заработную плату за месяц на количество рабочих дней месяца. Разделим 35000 на 22 и получим приблизительно 1591 рубль в день. Далее умножим стоимость одного рабочего дня на количество затраченных и получим 259333 рубля.

## 6. 2 Экономическая выгода разработки

Для обеспечения бесплатного доступа к Telegram-боту для всех пользователей, финансовая модель будет основана на доходах от рекламных баннеров, партнерских интеграций, а также на продаже токенов. Пользователям без подписки будет предоставляться ограниченное количество токенов в сутки, в то время как подписчики смогут использовать бота без ограничений. Например, при размещении четырех баннеров и достижении 10 000 просмотров в день можно оценить потенциальную прибыль от рекламы. Ключевым показателем является CTR (Click-Through Rate), который определяет процент пользователей, перешедших по рекламным баннерам на сайт рекламодателя. Среднее значение CTR составляет около 1%. При стоимости клика в 0.50$, это принесет примерно 400 кликов в сутки, что эквивалентно 200$ ежедневной выручки. При текущем курсе доллара в 102,34 рубля, и с учетом 365 дней в году, годовой доход от рекламы составит приблизительно 7520282 рублей. Эта цифра значительно превышает затраты на разработку и поддержку бота, почти в 26 раз. Кроме того, доход от продажи токенов будет дополнительно увеличивать общую прибыль.

ЗАКЛЮЧЕНИЕ

По мере распространения смартфонов и роста популярности использования Telegram-ботов в повседневных задачах, а также нейронных сетей, разработка таких информационных систем выглядит все более привлекательной для запуска коммерческих проектов.

В ходе работы было реализовано приложение Telegram-бот, полностью соответствующее поставленным в первой главе целям, а также современным стандартам качества для обеспечения лучшего пользовательского опыта. Для достижения этой задачи были проанализированы существующие аналоги, выявлены их недостатки и потенциальные улучшения.

Результатом выпускной квалификационной работы является полноценный программный продукт, помогающий жителям или гостям г. Ростова-на-Дону качественно и быстро погружаться в историю города и узнавать много нового о его достопримечательностях и памятниках культуры, посредством общения с искусственным интеллектом, способным анализировать объект на фотографиях и рассказывать о них.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Никитин А. А, Лиманова Н. И. Процесс распознавания изображения нейронной сетью// «Молодой ученый» Международный научный журнал. – 2020. – №5. С.23-25. (дата обращения 26.10.2024)
2. PyTorch documentation [Электронный ресурс]. URL: <https://pytorch.org/docs/stable/index.html> (дата обращения 25.09.2024)
3. Scikit-learn [Электронный ресурс]. URL: <https://scikit-learn.org/stable/index.html> (дата обращения 25.09.2024)
4. TensorFlow [Электронный ресурс]. URL: <https://www.tensorflow.org/?hl=ru> (дата обращения 25.09.2024)
5. Keras [Электронный ресурс]. URL: <https://keras.io/about/> (дата обращения 25.09.2024)
6. SciPy [Электронный ресурс]. URL: <https://docs.scipy.org/doc/scipy/tutorial/index.html> (дата обращения 25.09.2024)
7. PYPL PopularitY of Programming Language [Электронный ресурс]. URL: <https://pypl.github.io/PYPL.html> (дата обращения 25.09.2024)
8. Документация alexnet [Электронный ресурс]. URL: <https://docs.exponenta.ru/deeplearning/ref/alexnet.html> (дата обращения 26.10.2024)
9. ResNet [Электронный ресурс]. URL: <https://pytorch.org/vision/main/models/resnet.html> (дата обращения 26.10.2024)
10. И.Р. Петрова, Р.Х. Фахртдинов, Сулейманова А.А., Разживин И.О., Фазулзянов А.Г. Методология объектно-ориентированного моделирования. Язык UML // Учебно-методическое пособие по курсу «Методы анализа и построения информационных систем». – 2018. – С.28-38. (дата обращения 26.11.2024)
11. Буч Г., Рамбо Д., Якобсон И. Язык UML. Руководство пользователя. 2-е изд. – М. – 2022. – C. 40. (дата обращения 19.12.2024)
12. UML Practical Guide – All you need to know about UML modeling   
    [Электронный ресурс]. URL: [https://www.visual-paradigm.com/guide/umlunified-modeling-language/uml-practical-guide/](https://vk.com/away.php?to=https%3A%2F%2Fwww.visual-paradigm.com%2Fguide%2Fumlunified-modeling-language%2Fuml-practical-guide%2F&utf=1) (дата обращения 29.01.2025)
13. Кибермедиана [Электронный ресурс]. URL: <https://www.cybermedian.com/ru/a-comprehensive-guide-to-uml-class-diagram/#Diagramma_klassov_-_primer_instrumenta_diagrammy> (дата обращения 29.01.2025)
14. Гатман, Алекс Дж. Разберись в Data Science: как освоить науку о данных и научиться думать как эксперт. – 2024. – С.250-252. (дата обращения 1.02.2025)
15. Головко В. А., Краснопрошин В.В. Нейросетевые технологии обработки данных. – 2017. – C. 34. (дата обращения 01.02.2025)
16. MathWorks [Электронный ресурс]. URL: <https://www.mathworks.com/help/deeplearning/ug/transfer-learning-using-alexnet.html> (дата обращения 3.02.2025)
17. Майтак Р. В., Основополагающая концепция технологического прорыва глубокой нейронной сети Alexnet в задаче распознавания изображений. – 2023. – C.6. (дата обращения 3.02.2025)
18. Андриянов Н. А, Андриянов Д. А. О важности аугментации данных при машинном обучении в задачах обработки изображений в условиях дефицита данных. – 2020. – C.1-2. (дата обращения 10.02.2025)
19. Albumentations: Fast and Flexible Image Augmentations [Электронный ресурс] / A. Buslaev [et al.] // Information. 2020 № 2 URL: <https://www.mdpi.com/2078-2489/11/2/125> (дата обращения 10.02.2025).
20. Bochkovskiy A., Wang C.Y., Liao H.Y.M. YOLOv4: Optimal speed and accuracy of object detection [Электронный ресурс] // arXiv preprint arXiv: 2004.10934. 2020 URL: https://arxiv.org/abs/2004.10934 (дата обращения 10.02.2025).
21. Алимагадов К. А., Умняшкин С. В. Аугментация данных на основе вейвлет-фильтрации при обучении нейронных сетей. – 2023. – C. 438. (дата обращения 10.02.2025).
22. Афанасьев А. Д., Афанасьева Ж. С. Классификация изображений с помощью сверточных нейронных сетей: шаг за шагом. – 2020. – С.26-27. (дата обращения 15.02.2025).
23. Сербиев Р.А., Березан Д.Г. Оценка качества распознавания объектов на тепловизионных изображениях с помощью нейронных сетей. – 2023. – С.2. (дата обращения 15.02.2025).
24. Елисеев А. И., Минин Ю. В., Кулаков Ю. В. Решение задач глубокого обучения с использованием фреймворков PyTorch и PyTorch Lightning. – 2023. – C. 25-27. (дата обращения 19.02.2025).
25. Джулли А., Пал С. Библиотека Keras – инструмент глубокого обучения. – 2018. – С.14 (дата обращения 20.02.2025)
26. Куликов И. В. Нейросети на Python. Основы ИИ и машинного обучения. – 2025. – С.190 (дата обращения 1.03.2025)
27. Николенко С., Кадурин А., Архангельская Е. Глубокое обучение. - 2018. — C. 177 (дата обращения 1.03.2025)
28. Машинное и глубокое обучение [Электронный ресурс]. URL: <https://deepmachinelearning.ru/docs/Neural-networks/Training-simplification/BatchNorm> (дата обращения 1.03.2025)
29. А.В. Гайер, А.В. Шешкус, Ю.С. Черышова Аугментация обучающей выборки «на лету» для обучения нейронных сетей\* - 2018. – С. 152 (дата обращения 10.03.2025)
30. Аугментация данных [Электронный ресурс]. URL: <https://storage.piter.com/upload/contents/978544611677/978544611677_p.pdf> (дата обращения 10.03.2025)
31. Безопасность жизнедеятельности при работе с компьютером [Электронный ресурс]. URL: <https://ele74197079.narod.ru/pk.pdf> (дата обращения 15.03.2025)
32. Н. С. Коновалова Проблема безопасности при работе с компьютером – 2020. - С. 173 (дата обращения 15.03.2025)
33. Влияние компьютера на здоровье человека [Электронный ресурс]. – URL: <https://subscribe.ru/archive/science.health.anatomya/200109/28080512.html>. (дата обращения 15.03.2025)

# ПРИЛОЖЕНИЕ А

def plot\_confusion\_matrix(model, dataloader, class\_names):

model.eval()

all\_preds = []

all\_labels = []

with torch.no\_grad():

for inputs, labels in dataloader:

inputs = inputs.to(device)

outputs = model(inputs)

\_, preds = torch.max(outputs, 1)

all\_preds.extend(preds.cpu().numpy())

all\_labels.extend(labels.cpu().numpy())

valid\_indices = [i for i, label in enumerate(all\_labels) if label is not None]

all\_labels = [all\_labels[i] for i in valid\_indices]

all\_preds = [all\_preds[i] for i in valid\_indices]

cm = confusion\_matrix(all\_labels, all\_preds)

cm\_df = pandas.DataFrame(cm, index=class\_names, columns=class\_names)

plt.figure(figsize=(10, 8))

sns.heatmap(cm\_df, annot=True, fmt='d', cmap='Blues')

plt.title('Матрица ошибок')

plt.ylabel('Истинные классы')

plt.xlabel('Предсказанные классы')

plt.show()

# ПРИЛОЖЕНИЕ Б

@bot.message\_handler(content\_types=['photo'])

def repeat\_all\_messages(message):

language = user\_languages.get(message.chat.id, 'ru')

try:

# Получаем содержимое фото

file\_content = get\_photo(message)

# Преобразуем байты в изображение

image = byte2image(file\_content)

image = transform(image)

# Подготавливаем модель к инференсу

model.eval()

image = torch.unsqueeze(image, 0)

# Получаем предсказание

outputs = model(image)

\_, preds = torch.max(outputs, 1)

landmark\_name = classes[int(preds)]

print(f'Распознанная достопримечательность: {landmark\_name}') # Отладочное сообщение

logger.info(f'Распознанная достопримечательность: {landmark\_name}') # Логируем распознанное название

# Проверяем, является ли объект одной из 17 достопримечательностей

if landmark\_name in classes:

# Получаем язык пользователя

#language = user\_languages.get(message.chat.id, 'ru') # По умолчанию русский

# Проверяем, есть ли информация о достопримечательности

if language == 'ru':

info = landmark\_info\_ru.get(landmark\_name)

coordinates = landmark\_coordinates.get(landmark\_name) # Определяем координаты объектов

if info is None or coordinates is None:

bot.send\_message(message.chat.id, 'Извините, я не смог распознать эту достопримечательность.')

return

else:

info = landmark\_info\_en.get(landmark\_name)

coordinates = landmark\_coordinates.get(landmark\_name)

if info is None or coordinates is None:

bot.send\_message(message.chat.id, 'Sorry, I could not recognize this landmark.')

return

# Создаем ссылку на карту

map\_link = get\_map\_link(coordinates)

# Если информация найдена, отправляем сообщение

bot.send\_message(message.chat.id, text=f'{landmark\_name}\n\n{info}\n\n{map\_link}')

# Отправляем группу фотографий

media\_group = []

if landmark\_name in landmark\_photos:

for image\_link in landmark\_photos[landmark\_name]:

media\_group.append(telebot.types.InputMediaPhoto(image\_link))

try:

bot.send\_media\_group(message.chat.id, media\_group)

except Exception as e:

print(f'Ошибка при отправке изображения: {e}')

time.sleep(2)

# Запрашиваем подтверждение от пользователя

if language == 'ru':

bot.send\_message(message.chat.id, "Я ответил на ваш вопрос? (да/нет)")

else:

bot.send\_message(message.chat.id, "Have I answered your question? (yes/no)")

bot.register\_next\_step\_handler(message, process\_confirmation, image, landmark\_name)

else:

# Объект не является одной из 17 достопримечательностей

language = user\_languages.get(message.chat.id, 'ru') # По умолчанию русский

if language == 'ru':

bot.send\_message(message.chat.id, 'Извините, я не распознаю этот объект.')

else:

bot.send\_message(message.chat.id, 'Sorry, I do not recognize this object.')

except Exception as e:

logger.error(f'Ошибка: {e}', exc\_info=True) # Логируем ошибку с трассировкой

language = user\_languages.get(message.chat.id, 'ru') # По умолчанию русский

if language == 'ru':

bot.send\_message(message.chat.id, 'Упс, что-то пошло не так :(\nОбратитесь в службу поддержки!')

else:

bot.send\_message(message.chat.id, 'Oops, something went wrong :(\nPlease contact support!')

# ПРИЛОЖЕНИЕ В

def process\_incorrect\_data(message, image):

# Получаем правильное название от пользователя

correct\_landmark = message.text.strip().lower()

user\_id = message.chat.id

language = user\_languages.get(message.chat.id, 'ru')

# Увеличиваем счетчик для пользователя

if user\_id not in user\_counters:

user\_counters[user\_id] = 0

user\_counters[user\_id] += 1

if correct\_landmark in ['я не знаю', "i don't know"]:

# Формируем уникальное имя файла

timestamp = int(time.time()) # Используем временную метку

image\_path = f'incorrect\_data/{user\_id}\_{user\_counters[user\_id]}\_unknown.jpg'

# Сохраняем изображение

save\_image(image, image\_path)

else:

# Заменим пробелы на нижние подчеркивания, чтобы избежать проблем с именами файлов

correct\_landmark = correct\_landmark.replace(' ', '\_')

# Сохраняем изображение с правильным названием

image\_path = f'incorrect\_data/{user\_id}\_{user\_counters[user\_id]}\_{correct\_landmark}.jpg' # Формируем путь с названием

save\_image(image, image\_path) # Функция для сохранения изображения

# Сохраняем метку в текстовом файле или базе данных

with open('incorrect\_labels.txt', 'a') as f:

f.write(f'{image\_path},{correct\_landmark}\n')

bot.send\_message(message.chat.id,

'Спасибо! Данные сохранены.' if language == 'ru'

else 'Thank you! Data has been saved.')