МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет компьютерных технологий и прикладной математики
Кафедра анализа данных и искусственного интеллекта

КУРСОВАЯ РАБОТА

РАЗРАБОТКА WEB-СЕРВИСА ДЛЯ РАСПОЗНАВАНИЯ ОБЪЕКТОВ НА ИЗОБРАЖЕНИИ

Работу выполнил _____ (подпись) В.И. Алтухов

Направление подготовки 09.03.03 Прикладная информатика курс 3

Направленность (профиль) Прикладная информатика в экономике

Научный руководитель
ст. преподаватель _____ (подпись) Е.В. Казаковцева

Нормоконтролер
канд. физ.-мат. наук, доц. _____ (подпись) Г.В. Калайдина

Краснодар
2023

РЕФЕРАТ

Курсовая работа из 48 страниц, 32 рисунка и 9 источников.

WEB-ПРИЛОЖЕНИЕ, REACT.JS, JAVASCRIPT, FASTAPI, PYTHON, YOLO, ПОЛЬЗОВАТЕЛЬ, ПРИЛОЖЕНИЕ

Были изучены теоретический фундамент разработки web-приложения, подбирались инструменты и осуществлялось дальнейшее внедрение web-приложения при помощи React.js, FastAPI, YOLO.

Цель курсовой работы – разработать веб-приложение с внедренной моделью машинного обучения на примере сайта, распознающего кошек и собак на картинках.

Задачи:

– ознакомиться с теоретическими основами разработки web-приложений;

– ознакомиться с теоретическими основами разработки web-приложений;

– изучить подходящие инструменты для разработки web-приложений;

– выбрать датасет и архитектуру сети,

– изучить основы React.js, FastAPI, Object detection;

– создать полноценное web-приложение.

Объект исследования – web-приложение. Предметы исследования – React.js, FastAPI, Tensorlow, модель свёрточной нейронной сети. Итог проделанной работы – полноценное web-приложение.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Веб-технологии – это инструменты и методы, которые обеспечивают связь и взаимодействие между различными устройствами через Интернет. Веб-браузер – это программное обеспечение, которое отображает веб-страницы, представляющие собой документы, содержащие текст, изображения, видео и другие мультимедийные элементы. Веб-страницы связаны между собой гиперссылками, которые позволяют пользователям переходить с одной страницы на другую. Они написаны на языке разметки под названием HTML, который определяет структуру и содержание документа. Веб-страницы также могут использовать другие языки, такие как CSS и JavaScript, для улучшения внешнего вида и функциональности документа.

Веб-технологии можно разделить на две категории: frontend и backend. Интерфейсные технологии – это те, которые имеют дело с пользовательским интерфейсом и представлением веб-страницы, такие как HTML, CSS и JavaScript. Серверные технологии – это те, которые отвечают за обработку и хранение данных веб-страницы, такие как PHP, Python и SQL. Серверные технологии взаимодействуют с интерфейсными технологиями через протокол, называемый HTTP, который определяет способ обмена сообщениями между веб-сервером и веб-браузером.

Веб-технологии имеют множество применений и преимуществ для различных областей и отраслей промышленности. Они позволяют пользователям получать доступ к информации, услугам и развлечениям из любого места и в любое время. Они также способствуют сотрудничеству, общению и творчеству между людьми по всему миру. Веб-технологии постоянно развиваются и совершенствуются, чтобы соответствовать потребностям и ожиданиям пользователей и разработчиков.

Цель работы - разработать web-приложение по детекции рыбок на изображении

Задачи:

– ознакомиться с теоретическими основами разработки web-приложений;

– изучить подходящие инструменты для разработки web-приложений;

– выбрать датасет и архитектуру сети;

– изучить основы React.js, FastAPI, Object detection;

– создать полноценное web-приложение.

Курсовая работа состоит из трех глав, введения, заключения, списка использованной литературы и реферата. Первая глава работы посвящена теоретическим основам разработки веб-приложений. Во второй главе приведены сведения об архитектурах нейронных сетей. Третья глава посвящена описанию фронтенд, бэкенд частей, а также последующим внедрением YOLO в приложение.

1 WEB-технологии

1.1 Frontend

Для создания user-friendly интерфейс используется CSS и HTML. С их помощью можно создать только визуальную часть, функционал с ее помощью создать нельзя. Функционал пишется с помощью JavaScript, который может отлавливать события кнопок, принимать и обрабатывать данные, введенные пользователем. До появления фреймворков для небольшого сайта писали сотни строк кода на трех перечисленных технологиях. Появлегние первого фреймворка сделало работу программистов удобнее и качественней.

Для реализации своего проекта был выбран фреймвор React JS. Он является одной из самой востребованной технологией в веб индустрии.

React JS – это JavaScript-библиотека для создания пользовательских интерфейсов. React JS позволяет создавать одностраничные приложения, используя компонентный подход. Компоненты React JS – это переиспользуемые элементы интерфейса, которые могут принимать различные данные и реагировать на изменения состояния. React JS облегчает разработку и поддержку сложных интерфейсов, так как он отделяет логику от представления и использует виртуальный DOM для оптимизации рендеринга.

Для начала работы с React JS необходимо установить Node.js и запустить команду npx create-react-app в терминале, чтобы создать новое React-приложение. Затем можно перейти в папку с приложением и запустить команду npm start, чтобы запустить приложение в браузере. React-приложение состоит из файла index.html, который подключает скрипт с React-кодом, и файла index.js, который импортирует React-библиотеку и рендерит корневой компонент в элемент с id "root". Компоненты React JS могут быть определены как функции или классы, которые возвращают JSX.

JSX (JavaScript XML) – это расширение синтаксиса JavaScript, используемое в React.js для описания структуры пользовательского

интерфейса. JSX предоставляет возможность создавать древовидную структуру компонентов, которая в конечном итоге будет преобразована в обычный JavaScript-код, который React может интерпретировать и отобразить на странице.

В целом, JSX является мощным инструментом, который делает разработку пользовательского интерфейса с использованием React.js более простой и понятной. Он предоставляет удобный способ описания структуры компонентов и улучшает читаемость и поддерживаемость кода.

Node.js – это среда выполнения JavaScript, основанная на движке V8, разработанном Google. Она позволяет выполнять JavaScript-код на сервере, что открывает возможности для разработки полноценных веб-приложений с использованием React.js как на клиентской, так и на серверной стороне.

Node.js обладает внушительным функционалом. Например, он предоставляет возможность разрабатывать серверную часть приложения с использованием JavaScript. Это позволяет использовать React.js для создания как клиентской, так и серверной частей приложения, обеспечивая единый язык программирования на всем стеке разработки. С помощью Node.js можно создавать API-серверы, обрабатывать запросы от клиентской части React-приложения, взаимодействовать с базами данных, управлять авторизацией и многое другое. Для управление зависимостями и сборкой. Node.js вместе с npm (Node Package Manager) или Yarn предоставляет мощные инструменты для управления зависимостями и сборкой React-проекта. С помощью пакетного менеджера можно устанавливать и обновлять необходимые пакеты и модули. Скрипты сборки и тестирования могут быть настроены в файле package.json, что упрощает автоматизацию процесса разработки.

В целом, Node.js предоставляет обширные возможности для разработки серверной части приложений React.js, позволяя использовать JavaScript на всем стеке разработки. Это позволяет создавать эффективные и масштабируемые веб-приложения, объединяя клиентскую и серверную логику на единой платформе.

Axios – это библиотека JavaScript, которая используется для выполнения HTTP-запросов из браузера или среды выполнения Node.js. В React.js Axios широко используется для взаимодействия с внешними API и отправки запросов на сервер.

Подробнее о возможностях Axios в React.js:

1) Установка и импорт Axios. Для использования Axios в React.js необходимо сначала установить его с помощью пакетного менеджера, такого как npm или Yarn. После установки Axios может быть импортирован в компоненты React.js для использования.

2) Отправка HTTP-запросов. Axios предоставляет удобные методы для отправки различных типов HTTP-запросов, таких как GET, POST, PUT, DELETE и другие. Метод axios.request(config) используется для отправки настраиваемого запроса с заданными параметрами. Методы axios.get(url, config), axios.post(url, data, config) и другие предоставляют упрощенный синтаксис для отправки определенных типов запросов.

3) Обработка ответов. Axios возвращает обещания (Promises) или использует синтаксис async/await для обработки ответов на запросы. Обещания позволяют выполнять действия после завершения запроса, например, обновление состояния компонента или обработка полученных данных. В случае успешного выполнения запроса, Axios возвращает объект ответа, содержащий статус, заголовки и тело ответа. В случае ошибки, Axios генерирует исключение, которое можно обработать с помощью блока catch или использования метода .catch().

4) Передача данных. Axios позволяет отправлять данные вместе с запросами, такие как параметры URL, данные формы или тело запроса. Для передачи параметров в URL используется параметр params. Для отправки данных в теле запроса используется параметр data.

5) Интерцепторы. Axios позволяет использовать интерцепторы для перехвата и изменения запросов или ответов перед их отправкой или обработкой. Интерцепторы могут быть использованы для добавления

заголовков, обработки ошибок, добавления авторизации и других задач. Интерцепторы позволяют создавать глобальные настройки для всех запросов или специфические настройки для отдельных запросов.

Axios облегчает выполнение HTTP-запросов в React.js и предоставляет удобные методы для обработки ответов и передачи данных. Он является популярным выбором для взаимодействия с сервером и внешними API в приложениях React.js.

Axios я использовал для отправки изображения на сервер и принятия ответа.

useState – это хук (hook) в React.js, который позволяет функциональным компонентам React сохранять состояние и обновлять его. Он предоставляет возможность использовать состояние внутри функциональных компонентов, что ранее было доступно только в классовых компонентах.

UseState также поддерживает импорт и использование хука. Чтобы использовать useState, его необходимо импортировать из библиотеки React. Хук useState используется внутри функциональных компонентов с помощью вызова, который возвращает массив с двумя элементами: текущее состояние и функцию для его обновления. UseState может инициализировать состояние компонента. Начальное значение может быть передано в качестве аргумента useState. UseState также поддерживает чтение состояния, обратившись к переменной, которая была задана в useState. Еще одним функционалом является обновление состояния. Для этого используется функция, которая была возвращена хуком useState. Вызов функции обновления изменяет состояние компонента и вызывает его повторное рендеринг. Функция обновления может быть вызвана с новым значением состояния или функцией, которая принимает предыдущее состояние и возвращает новое. В некоторых случаях может потребоваться обновить состояние, исходя из предыдущего значения. useState поддерживает функциональное обновление. Функция обновления получает предыдущее состояние и возвращает новое состояние.

Хук useState предоставляет простой способ управления состоянием в функциональных компонентах React. Он позволяет сохранять и обновлять значения состояния, что делает функциональные компоненты более гибкими и мощными.

1.2 Backend. FastApi

FastAPI – это современный веб-фреймворк для создания API-интерфейсов с помощью языка программирования Python, который был разработан, как быстрый, простой в использовании и высоко масштабируемый, что делает его идеальным выбором для создания высокопроизводительных веб-приложений.

Ключевые функции FastAPI:

1) Производительность. Он построен поверх мощной спецификации ASGI (Asynchronous Server Gateway Interface), позволяющая обрабатывать большое количество запросов одновременно.

Одной из оптимизаций является использование асинхронного программирования с библиотекой asyncio на Python. Это позволяет FastAPI обрабатывать несколько запросов одновременно, не блокируя основной поток. Асинхронное программирование может значительно повысить производительность веб-приложений, особенно при работе с операциями ввода-вывода, такими как запросы к базе данных. FastAPI также использует Pydantic, библиотеку проверки данных и сериализации, для оптимизации обработки данных. Pydantic использует пользовательскую модель данных, определенную разработчиком, что может помочь снизить накладные расходы на обработку данных и повысить производительность.

Кроме того, FastAPI использует сервер uvicorn, который является высокопроизводительным сервером, оптимизированным для обработки большого количества запросов. Uvicorn использует асинхронный ввод-вывод

для быстрой и эффективной обработки запросов. FastAPI также предоставляет несколько встроенных инструментов мониторинга производительности, таких как возможность регистрировать запросы и ответы, а также профилировать производительность с помощью таких инструментов, как cProfile или pyinstrument.

В целом, ориентация FastAPI на производительность делает его отличным выбором для разработчиков, которым требуются высокоскоростные API с низкой задержкой и высокой пропускной способностью.

2) Аннотация типов. FastAPI использует подсказки типов, обеспечивающие богатый и выразительный интерфейс для определения конечных точек API. Это позволяет легко писать хорошо документированный код, который прост в понимании и обслуживании.

Аннотации типов в FastAPI относятся к процессу явного объявления типов аргументов функции, возвращаемых значений и атрибутов класса в коде Python. FastAPI использует аннотации типов для обеспечения типобезопасности и более выразительной разработки API. FastAPI использует модуль ввода на Python для обеспечения поддержки аннотаций типов. Он поддерживает широкий спектр типов, включая простые типы, такие как str, int, bool и т.д., а также более сложные типы, такие как списки, кортежи и словари. Кроме того, FastAPI также поддерживает пользовательские типы, которые могут быть определены с помощью модуля ввода Python или пользовательских классов.

FastAPI использует аннотации типов для создания подробной документации для каждой конечной точки API. Эта документация включает информацию об ожидаемых типах данных для каждого параметра запроса и ответа, что облегчает разработчикам понимание и использование API.В целом, аннотации типов являются мощной функцией FastAPI, которая позволяет разработчикам писать более безопасный, выразительный и эффективный код. Они обеспечивают автоматическую проверку и преобразование данных, автоматическое создание документации по API и расширенную поддержку

IDE, что делает FastAPI мощным и эффективным инструментом для создания современных API.

3) Pydantic – это мощная библиотека для проверки данных, гарантирующий, что данные, передаваемые конечным точкам API и из них, являются действительными и соответствуют ожидаемой схеме. Pydantic разработан для бесперебойной работы с FastAPI и предоставляет ряд функций, которые позволяют разработчикам создавать эффективные и масштабируемые веб-приложения. Pydantic предоставляет набор инструментов для определения моделей данных, включая классы, которые определяют структуру данных, и средства проверки, которые гарантируют, что данные находятся в правильном формате. Pydantic также предоставляет встроенную поддержку JSON, YAML и других форматов данных, что упрощает работу с данными из разных источников. Одной из ключевых особенностей Pydantic является его способность автоматически генерировать документацию API на основе определенных моделей данных. Это позволяет разработчикам легко создавать понятную и всеобъемлющую документацию для своих API без необходимости написания дополнительного кода. В дополнение к своим функциям проверки данных и сериализации, Pydantic также предоставляет ряд других полезных инструментов, включая поддержку пользовательских валидаторов и валидаторов, которые могут быть определены с помощью регулярных выражений. Pydantic также включает в себя мощный конструктор запросов, который позволяет разработчикам легко создавать сложные запросы на основе своих моделей данных. В целом, Pydantic – это мощный инструмент, который упрощает разработчикам работу со сложными моделями данных в приложениях FastAPI. Его функции проверки данных и сериализации делают его ценным дополнением к любому инструментарию веб-разработки, а поддержка автоматической генерации документации API делает его любимым среди многих разработчиков.

4) Асинхронное программирование, что является ключевой функцией для создания высокопроизводительных веб-приложений. Это

позволяет разработчикам писать асинхронный код, используя синтаксис async/await в Python. FastAPI известен своей исключительной производительностью благодаря своим асинхронным возможностям.

Платформа FastAPI изначально поддерживает асинхронное программирование и использует асинхронные возможности Python, такие как async /await и модуль asyncio. Используя асинхронное программирование, FastAPI может обрабатывать большее количество запросов одновременно и эффективно, что приводит к повышению производительности.

FastAPI использует стандарт ASGI (Asynchronous Server Gateway Interface) для связи между веб-сервером и приложением. ASGI – это более современный и эффективный стандарт, чем традиционный WSGI (интерфейс шлюза веб-сервера), используемый большинством веб-фреймворков Python.

FastAPI также поддерживает использование синтаксиса async и await для определения маршрутов и функций, что позволяет разработчикам писать асинхронный код более естественно. Эта функция позволяет разработчикам писать эффективный код, выполняя неблокирующие операции ввода-вывода, такие как запрос к базе данных, вызов API или выполнение длительных вычислений.

Кроме того, FastAPI предоставляет дополнительные инструменты, помогающие разработчикам в асинхронном программировании. Одним из наиболее примечательных является встроенная поддержка библиотеки Pydantic, которая позволяет разработчикам объявлять структуру входящих запросов и исходящих ответов типобезопасным способом. Эта функция помогает обнаруживать ошибки типа на ранней стадии и предотвращает исключения во время выполнения, что приводит к повышению надежности кода.

В целом, поддержка асинхронного программирования FastAPI позволяет разработчикам создавать высокопроизводительные API, которые могут обрабатывать большое количество запросов одновременно.

5) Поддержка SQL и NoSQL. Она упрощает создание приложений, требующих постоянного хранения и извлечения данных, и позволяет разработчикам выбирать базу данных, которая наилучшим образом соответствует их потребностям. Главный инструмент для работы с базами данных – SQLAlchemy. FastAPI – это веб-фреймворк на Python, который поддерживает различные базы данных. Он обеспечивает легкую интеграцию с базами данных за счет использования ORMs (объектно-реляционное сопоставление). ORM сопоставляют классы Python с таблицами в базе данных, упрощая работу с базами данных. FastAPI поддерживает несколько популярных баз данных, таких как PostgreSQL, MySQL, SQLite и Microsoft SQL Server. Он обеспечивает поддержку этих баз данных с использованием различных ORM, таких как SQLAlchemy, Tortoise-ORM и GINO.

6) SQLAlchemy – это популярный ORM, который поддерживает множество различных систем баз данных. Он обеспечивает высокий уровень абстракции по сравнению с системами баз данных и позволяет разработчикам писать запросы к базе данных на Python. SQLAlchemy предоставляет поддержку для создания таблиц, вставки данных и запроса данных.

Tortoise-ORM – это еще один ORM, созданный специально для асинхронных приложений. Это простой в использовании ORM, который обеспечивает поддержку создания таблиц, вставки данных, запроса данных и многого другого. Tortoise-ORM также обеспечивает поддержку миграции баз данных, упрощая управление изменениями в схеме базы данных.

GINO – это легкий ORM, предназначенный для работы с асинхронными приложениями. Он обеспечивает поддержку баз данных PostgreSQL, MySQL и SQLite. GINO обеспечивает высокий уровень абстракции по сравнению с системами баз данных и позволяет разработчикам писать запросы к базе данных на Python. FastAPI предоставляет простой в использовании интерфейс для интеграции с этими ORMS. Фреймворк предоставляет декораторы для определения конечных точек, которые взаимодействуют с базами данных. Декоратор @app.on_event("startup") можно использовать для инициализации

подключений к базе данных при запуске приложения. Аналогично, декоратор @app.on_event("завершение работы") можно использовать для закрытия подключений к базе данных при завершении работы приложения.

В дополнение к ORMs, FastAPI также предоставляет поддержку необработанных SQL-запросов. Это позволяет разработчикам при необходимости писать пользовательские запросы на SQL.

В целом, FastAPI предоставляет комплексную систему интеграции баз данных, которая упрощает работу с базами данных в приложениях на Python.

Высокая производительность и масштабируемость FastAPI также делают его идеальным выбором для создания приложений машинного обучения. Фреймворк предоставляет ряд встроенных функций для работы с моделями машинного обучения, включая поддержку асинхронной обработки и обновлений в режиме реального времени. Это позволяет создавать мощные и масштабируемые приложения для машинного обучения, способные обрабатывать большие объемы данных, выдавать быстрые и точные результаты.

2 Нейронные сети

2.1 Общая теория нейронных сетей

Нейронные сети – это компьютерные модели, вдохновленные работой человеческого мозга. Они представляют собой сеть взаимосвязанных и взаимодействующих искусственных нейронов, которые обрабатывают и передают информацию. Такие сети имеют широкий спектр применений, включая распознавание образов, анализ текста, прогнозирование временных рядов, управление роботами и многое другое.

Основные компоненты нейронной сети включают нейроны, веса, функции активации и архитектуру сети. Давайте рассмотрим каждый из них подробнее:

1) Нейроны – основные строительные блоки нейронной сети. Они имитируют работу нейронов в человеческом мозге. Они могут принимать входные сигналы, выполнять вычисления и генерировать выходной сигнал. Входные сигналы могут быть числами или векторами, и каждый вход связан с весом, который определяет его важность для вычислений нейрона. Нейрон также имеет смещение (bias), которое добавляется к взвешенной сумме входных сигналов. Нейроны объединяются в слои, и информация передается от одного слоя к другому.

2) Вес – ассоциированная связь между нейронами. Он определяет важность входного сигнала для вычислений, выполняемых нейроном. Во время обучения нейронной сети веса настраиваются таким образом, чтобы минимизировать ошибку или достичь требуемого выхода. Изначально веса могут быть случайно инициализированы, а затем обновляются в процессе обучения, используя методы оптимизации, такие как градиентный спуск.

3) Функции активации - функции, которые определяют выходной сигнал нейрона на основе его входа. Они добавляют нелинейность в сеть и

позволяют ей моделировать сложные отношения между входными и выходными данными. Без нелинейности нейронная сеть была бы ограничена линейными преобразованиями, и ее способность моделировать сложные функции была бы сильно ограничена. Некоторые популярные функции активации включают сигмоиду, гиперболический тангенс, ReLU (Rectified Linear Unit), Leaky ReLU и softmax.

Комбинация этих компонентов - нейронов, весов, функций активации - определяет способность нейронной сети моделировать сложные отношения, обнаруживать закономерности в данных и выполнять задачи, такие как классификация, регрессия, сегментация и генерация. Каждый компонент имеет свою роль в процессе работы нейронной сети, и их правильное настройка и комбинация играют важную роль в достижении высокой производительности сети.

Процесс обучения нейронной сети – это итеративный процесс, в ходе которого сеть настраивает свои веса и параметры, чтобы минимизировать ошибку и улучшить свою производительность. Обычно обучение нейронной сети включает следующее:

1) Инициализация весов: Начальные значения весов в нейронной сети обычно устанавливаются случайным образом, хотя существуют и другие методы инициализации весов. Это важный шаг, так как исходные значения весов влияют на начальное поведение сети.

2) Прямое распространение (Forward Propagation): во время прямого распространения данные передаются через сеть от входного слоя к выходному слою. Каждый нейрон вычисляет свой выход на основе входных сигналов и соответствующих весов, используя функцию активации. Выход последнего слоя нейронной сети сравнивается с ожидаемыми значениями для вычисления ошибки.

3) Расчет ошибки (Loss Calculation): Ошибка определяет, насколько предсказанные значения отличаются от фактических значений. Для разных типов задач используются различные функции потерь, такие как

среднеквадратичная ошибка (MSE) для задач регрессии или перекрестная энтропия (Cross-Entropy) для задач классификации.

4) Обратное распространение ошибки (Backpropagation): Обратное распространение – это процесс, при котором ошибка сети распространяется от выходного слоя к входному. Ошибка каждого нейрона вычисляется на основе его вклада в общую ошибку сети. Затем градиенты ошибки передаются обратно через сеть, чтобы определить, как изменение весов каждого нейрона влияет на ошибку.

5) Обновление весов (Weight Update): после вычисления градиентов сеть обновляет веса, чтобы уменьшить ошибку. Это обычно осуществляется с использованием алгоритма оптимизации, такого как стохастический градиентный спуск (SGD) или его вариации. Алгоритмы оптимизации определяют, какие изменения в весах должны быть внесены на основе градиентов ошибки и гиперпараметров, таких как скорость обучения (learning rate).

6) Итерации: Процессы прямого и обратного распространения ошибки повторяются на протяжении нескольких эпох обучения. Одна эпоха представляет собой один полный проход по всем обучающим данным. Чем больше итераций и эпох, тем больше сеть имеет возможность улучшить свою производительность и обобщающую способность.

7) Остановка обучения: Обучение может быть остановлено, когда достигнуты заданные критерии остановки, например, максимальное количество эпох или достаточное уменьшение ошибки. Также можно использовать валидационный набор данных для оценки производительности сети на новые данные и принятия решения о прекращении обучения. Процесс обучения нейронной сети требует настройки нескольких гиперпараметров, таких как скорость обучения, количество слоев и нейронов в сети, функции активации и другие. Подбор оптимальных гиперпараметров является важным аспектом обучения нейронной сети и может быть выполнен с использованием

методов оптимизации, таких как поиск по сетке (grid search) или случайный поиск (random search).

Они подходят для решения многих задач:

– Классификация. Когда необходимо присвоить исследуемому объект к какому-либо классу;

– Предсказание. Когда нужно предсказать какой будет результат для исследуемых данных;

– Распознавание. Когда нужно определить нужный объект на фото, видео, аудиозаписи.

Виды нейронных сетей:

1) Нейронные сети прямого распространения (feed forward neural networks, FF или FFNN) и перцептроны (perceptrons, P) (Рисунок 1) являются двоольно линейными, которые передают информацию от входного слоя к слою выхода, где каждый слой состоит из входных, скрытых или выходных нейронов. Нейроны принадлежащие, общему слою не связаны между собой, а соседние слои зачастую полностью связаны. Простейшая нейронная сеть имеет два нейрона на входе и один на выходе, и используется в качестве модели логических вентилей. FFNN зачастую обучается по алгоритму обратного распространения ошибки, где нейронная сеть получает большое количество входных и выходных данных. Такой подход называется обучение с учителем, главное отличие от обучения без учителя заключает в том, что выходные данные нейронная сеть вычисляет самостоятельно. При достаточном количестве скрытых нейронов, нейронная сеть способна смоделировать взаимодействие между данными на входных и выходных слоях.

Рисунок 1 – Нейронные сети прямого распространения и перцептроны

```python
class FeedforwardNeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.weights1 = np.random.randn(input_size, hidden_size)
        self.bias1 = np.random.randn(hidden_size)
        self.weights2 = np.random.randn(hidden_size, output_size)
        self.bias2 = np.random.randn(output_size)

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def forward(self, x):
        hidden_layer = self.sigmoid(np.dot(x, self.weights1) + self.bias1)
        output_layer = np.dot(hidden_layer, self.weights2) + self.bias2
        return output_layer
```

Рисунок 2 – Код нейронной сети прямого распространения и перцептрона

2) Сети радиально-базисных функций (radial basis function, RBF) – это тип нейронной сети, которая широко используется для задач аппроксимации функций и классификации. RBFN – это сеть с прямой связью, которая состоит из трех уровней: входного уровня, скрытого уровня и выходного уровня. Скрытый уровень сети использует радиальные базисные функции (RBFs) для вычисления своих активаций, которые затем используются для вычисления выходных данных сети.

RBFN особенно полезен для аппроксимации функций, которые имеют нелинейную зависимость между входными и выходными переменными. Сеть достигает этого, используя набор RBFS для сопоставления входного пространства с пространством более высокой размерности, где

взаимосвязь между входными и выходными переменными линейна. RBF центрируются в определенных точках входного пространства, и их ширина определяет степень их влияния на выходные данные сети. (Рисунок 3).

Рисунок 3 – Сети радиально-базисных функций

```python
class RadialBasisFunctionNetwork:
    def __init__(self, input_size, hidden_size, output_size, sigma):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.sigma = sigma
        self.centers = np.random.randn(hidden_size, input_size)
        self.weights = np.random.randn(hidden_size, output_size)

    def radial_basis_function(self, x, c):
        return np.exp(-np.sum((x - c)**2) / (2 * self.sigma**2))

    def forward(self, x):
        hidden_layer = np.zeros((x.shape[0], self.hidden_size))

        for i in range(self.hidden_size):
            hidden_layer[:, i] = self.radial_basis_function(x, self.centers[i])

        return np.dot(hidden_layer, self.weights)
```

Рисунок 4 – Код сети радиально-базисной функции

3) Нейронная сеть Хопфилда (Hopfield network, HN) – Сеть Хопфилда – это тип рекуррентной нейронной сети (RNN), которая была впервые представлена Джоном Хопфилдом в 1982 году. Это форма ассоциативной памяти, означающая, что она способна изучать шаблоны и извлекать их из неполного или зашумленного ввода (Рисунок 5).

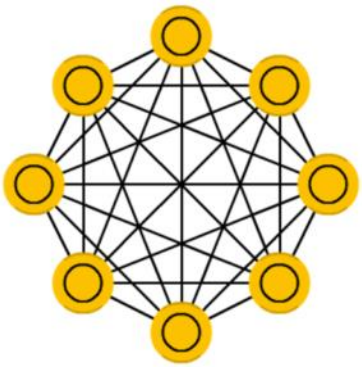


Рисунок 5 – Нейронная сеть Хопфилда

Сеть состоит из набора узлов или нейронов, каждый из которых может быть либо "включен", либо "выключен". Состояние каждого нейрона определяется взвешенной суммой состояний всех остальных нейронов в сети. Эти веса определяются на этапе обучения, когда сеть обучается сохранять набор "шаблонов памяти" путем корректировки весов на основе входных данных.

Сеть Хопфилда обладает рядом интересных свойств, включая динамику аттрактора, означающую, что она сходится к стабильному состоянию независимо от начальных входных данных, и ограничения пропускной способности, означающие, что количество шаблонов, которые могут быть сохранены, ограничено размером сети.

Несмотря на свою теоретическую привлекательность, сеть Хопфилда имеет некоторые ограничения на практике. Он в основном используется для задач распознавания образов и поиска информации и может справляться с более сложными проблемами, такими как классификация или регрессия. Кроме того, ограничения пропускной способности могут затруднить хранение большого количества шаблонов, а итеративный процесс обновления может быть медленным в больших сетях.

```python
class HopfieldNetwork:
    def __init__(self, n):
        self.n = n
        self.weights = np.zeros((n, n))

    def train(self, patterns):
        m = patterns.shape[0]
        self.weights = np.zeros((self.n, self.n))

        for i in range(m):
            pattern = patterns[i, :].reshape((-1, 1))
            self.weights += np.dot(pattern, pattern.T)

        self.weights /= m
        np.fill_diagonal(self.weights, 0)

    def energy(self, x):
        return -0.5 * np.dot(np.dot(x.T, self.weights), x)

    def update(self, x):
        x_new = np.sign(np.dot(self.weights, x))
        return x_new

    def predict(self, x, num_iterations=10):
        x_new = x

        for _ in range(num_iterations):
            x_new = self.update(x_new)

        return x_new
```

Рисунок 6 – Код нейронной сети Хопфилда

4) Цепь Маркова – это вероятностная модель, которая широко используется в различных областях, включая физику, экономику и информатику. Это случайный процесс, который следует свойству Маркова, что означает, что будущее состояние системы зависит только от текущего состояния и не зависит от прошлых состояний (**Ошибка! Источник ссылки не найден.**).

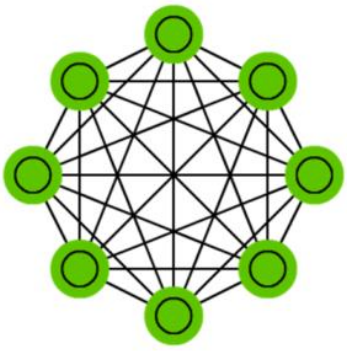
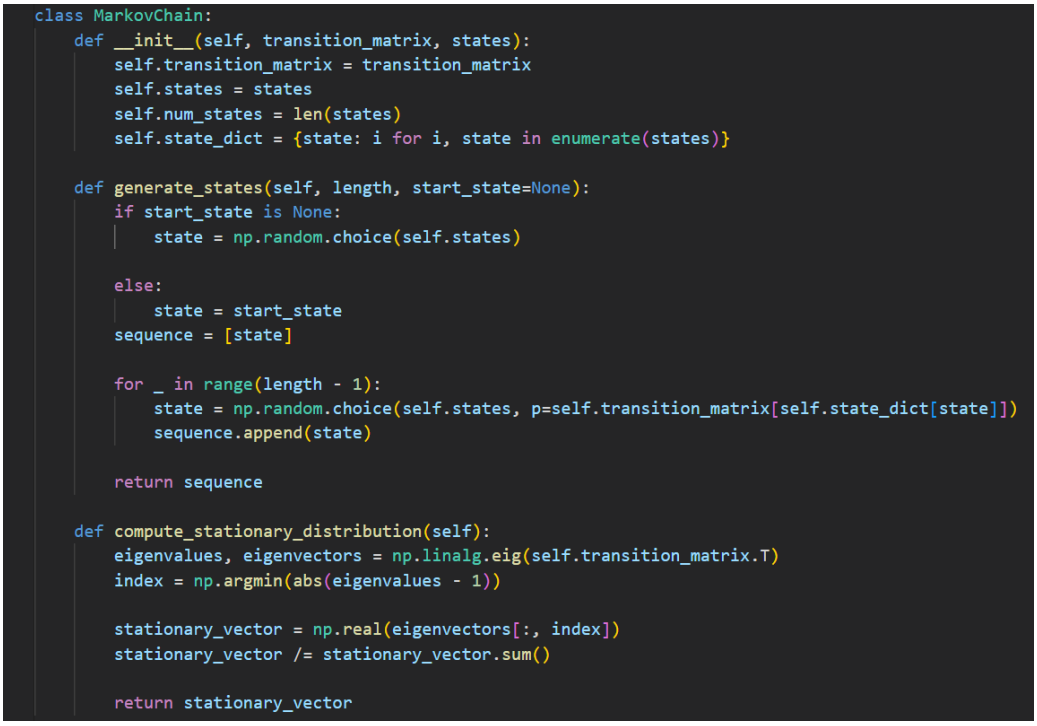


Рисунок 7 – Цепи Маркова



```python
class MarkovChain:
    def __init__(self, transition_matrix, states):
        self.transition_matrix = transition_matrix
        self.states = states
        self.num_states = len(states)
        self.state_dict = {state: i for i, state in enumerate(states)}

    def generate_states(self, length, start_state=None):
        if start_state is None:
            state = np.random.choice(self.states)

        else:
            state = start_state
        sequence = [state]

        for _ in range(length - 1):
            state = np.random.choice(self.states, p=self.transition_matrix[self.state_dict[state]])
            sequence.append(state)

        return sequence

    def compute_stationary_distribution(self):
        eigenvalues, eigenvectors = np.linalg.eig(self.transition_matrix.T)
        index = np.argmin(abs(eigenvalues - 1))

        stationary_vector = np.real(eigenvectors[:, index])
        stationary_vector /= stationary_vector.sum()

        return stationary_vector
```

Рисунок 8 – Код цепи Маркова

В контексте нейронных сетей цепочка Маркова может использоваться для моделирования последовательностей данных, таких как временные ряды или естественный язык. Основная идея заключается в представлении данных в виде последовательности дискретных состояний, где каждое состояние соответствует определенной функции или атрибуту данных. Затем сеть изучает набор вероятностей перехода между состояниями, которые определяют вероятность перехода из одного состояния в другое.

Одним из ключевых преимуществ нейронной сети с цепью Маркова является ее способность фиксировать долгосрочные зависимости в данных, что важно для таких задач, как языковое моделирование или распознавание речи. Это достигается за счет использования марковской модели высокого порядка, которая учитывает множество прошлых состояний при прогнозировании будущего состояния. Порядок построения модели может быть скорректирован в зависимости от сложности данных и объема доступных обучающих данных.

5) Машины Больцмана (BM) – это тип генеративной нейронной сети, которая способна изучать сложные распределения вероятностей по входным данным. Они основаны на принципах статистической механики и впервые были введены в конце 1980-х годов Хинтоном и Сейновски. (Рисунок 9).

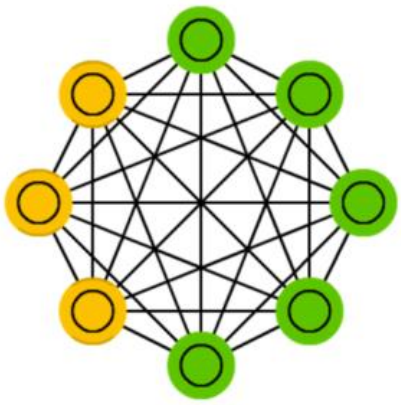


Рисунок 9 – Машина Больцмана

BM состоит из набора двоичных единиц, которые соединены друг с другом посредством взвешенных соединений. Каждый блок либо "включен" (со значением 1), либо "выключен" (со значением 0). Веса между блоками представляют прочность соединения между ними. BM обучаются с использованием процесса, называемого контрастивной дивергенцией, который включает в себя итеративное обновление весовых коэффициентов на основе разницы между входными данными и выходными данными модели. BM обладает рядом преимуществ, включая способность изучать сложные закономерности в данных и генерировать новые выборки, похожие на обучающие данные. Они также способны обрабатывать зашумленные или неполные данные, что делает их полезными в таких приложениях, как распознавание изображений и речи, а также обработка естественного языка.

```python
class BoltzmannMachine:
    def __init__(self, num_neurons):
        self.num_neurons = num_neurons
        self.weights = np.zeros((num_neurons, num_neurons))
        self.biases = np.zeros(num_neurons)

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def energy(self, state):
        return -0.5 * np.dot(state, np.dot(self.weights, state)) - np.dot(self.biases, state)

    def gibbs_step(self, state):
        energies = self.biases + np.dot(self.weights, state)
        probabilities = self.sigmoid(energies)
        new_state = np.random.binomial(1, probabilities)
        return new_state

    def train(self, data, learning_rate=0.1, num_epochs=1000):
        num_examples = data.shape[0]
        for epoch in range(num_epochs):
            for example in range(num_examples):
                state = data[example]
                new_state = self.gibbs_step(state)

                pos_associations = np.outer(state, state)
                neg_associations = np.outer(new_state, new_state)

                self.weights += learning_rate * \
                    (pos_associations - neg_associations)
                self.biases += learning_rate * (state - new_state)

    def predict(self, state):
        energies = self.biases + np.dot(self.weights, state)
        probabilities = self.sigmoid(energies)
        new_state = np.random.binomial(1, probabilities)
        return new_state
```

Рисунок 10 – Код машины Больцмана

6) Ограниченная машина Больцмана (restricted Boltzmann machine, RBM) – это тип генеративной нейронной сети, которая используется для обучения без присмотра. Это двухслойная нейронная сеть, состоящая из видимых и скрытых блоков. Видимые единицы представляют входные данные, в то время как скрытые единицы используются для изучения базовой структуры данных. (Рисунок 11).

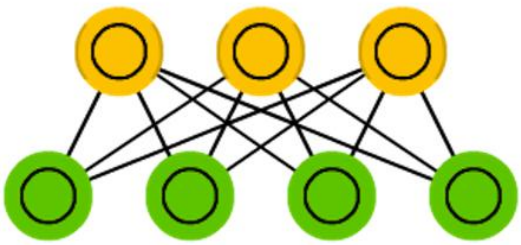


Рисунок 11 – Ограниченная машина Больцмана

```python
class RBM:
    def __init__(self, num_visible, num_hidden):
        self.num_visible = num_visible
        self.num_hidden = num_hidden
        self.weights = np.random.normal(0, 0.1, (num_visible, num_hidden))
        self.visible_bias = np.zeros(num_visible)
        self.hidden_bias = np.zeros(num_hidden)

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def energy(self, visible, hidden):
        return -np.dot(visible, self.visible_bias) - np.dot(hidden, self.hidden_bias) - np.dot(visible, np.dot(self.weights, hidden))

    def gibbs_step(self, visible):
        hidden_probabilities = self.sigmoid(np.dot(visible, self.weights) + self.hidden_bias)
        hidden_states = np.random.binomial(1, hidden_probabilities)

        visible_probabilities = self.sigmoid(np.dot(hidden_states, self.weights.T) + self.visible_bias)
        visible_states = np.random.binomial(1, visible_probabilities)

        return (visible_states, hidden_states)

    def train(self, data, learning_rate=0.1, num_epochs=1000):
        num_examples = data.shape[0]

        for epoch in range(num_epochs):
            for example in range(num_examples):
                visible = data[example]
                hidden_probabilities = self.sigmoid(np.dot(visible, self.weights) + self.hidden_bias)

                hidden_states = np.random.binomial(1, hidden_probabilities)

                positive_associations = np.outer(visible, hidden_probabilities)

                negative_visible, negative_hidden = self.gibbs_step(visible)
                negative_associations = np.outer(negative_visible, negative_hidden)

                self.weights += learning_rate *  (positive_associations - negative_associations)
                self.visible_bias += learning_rate * (visible - negative_visible)
                self.hidden_bias += learning_rate * (hidden_probabilities - negative_hidden)

    def predict(self, visible):
        hidden_probabilities = self.sigmoid(np.dot(visible, self.weights) + self.hidden_bias)
        hidden_states = np.random.binomial(1, hidden_probabilities)

        return hidden_states
```

Рисунок 12 – Код ограниченной машины Больцмана

RBM обучается с использованием алгоритма контрастной дивергенции, который является разновидностью стохастического градиентного спуска. Цель процесса обучения - узнать веса связей между видимыми и скрытыми единицами, которые максимизируют вероятность входных данных. Это достигается за счет минимизации ошибки восстановления, которая представляет собой разницу между входными данными и их восстановлением по скрытым блокам.

Одной из уникальных особенностей RBM является его способность усваивать сжатое представление входных данных. Это достигается за счет использования меньшего количества скрытых единиц, чем видимых, что вынуждает RBM изучать компактное представление входных данных.

7) Автокодировщик (autoencoder, AE) – это алгоритм нейронной сети, который широко используется в неконтролируемом обучении для уменьшения

размерности, изучения функций и сжатия данных. Нейронные сети автоэнкодера состоят из сети кодера, которая сопоставляет входные данные с представлением более низкой размерности, и сети декодера, которая восстанавливает исходные входные данные из представления более низкой размерности. (Рисунок 13).

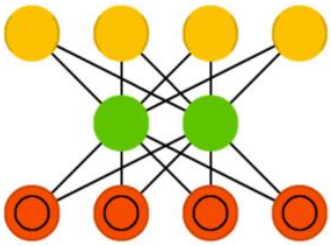


Рисунок 13 – Автокодировщик

Целью автоэнкодера является изучение сжатого представления входных данных, которое фиксирует наиболее важные характеристики данных, отбрасывая менее важные детали. Это сжатое представление может быть использовано для таких задач, как визуализация данных, обнаружение аномалий и генерация данных.

Кодирующая сеть автоэнкодера состоит из одного или нескольких скрытых слоев, которые применяют нелинейные преобразования к входным данным, постепенно уменьшая размерность данных, пока они не достигнут желаемого представления с меньшей размерностью. Сеть декодера выполняет противоположную операцию, принимая сжатое представление в качестве входных данных и постепенно восстанавливая исходные данные до тех пор, пока они не достигнут той же размерности, что и входные данные.

```python
class Autoencoder(tf.keras.Model):
    def __init__(self, input_dim, encoding_dim):
        super().__init__()

        self.input_layer = tf.keras.layers.Input(input_dim)
        self.hidden_layer = tf.keras.layers.Dense(encoding_dim, activation='relu')
        self.output_layer = tf.keras.layers.Dense(input_dim, activation='sigmoid')

    def model(self):
        input_layer = self.input_layer
        hidden_layer = self.hidden_layer(input_layer)
        output_layer = self.output_layer(hidden_layer)

        autoencoder = tf.keras.models.Model(inputs=input_layer, outputs=output_layer)
        return autoencoder
```

Рисунок 14 – Код автокодировщика

8) Разреженный автокодировщик (sparse autoencoder, SAE) – Разреженный автоэнкодер – это тип нейронной сети, который используется для неконтролируемого изучения объектов из немаркированных данных. Ключевая особенность разреженного автоэнкодера заключается в том, что он поощряет разреженность изучаемых функций, что означает, что только небольшое подмножество функций активируется для любого заданного ввода. Это может привести к более эффективному и надежному представлению входных данных. (Рисунок 15).

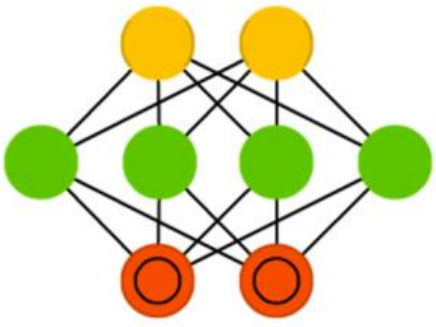


Рисунок 15 – Разреженный автокодировщик

Разреженный автоэнкодер работает путем предварительного кодирования входных данных в низкоразмерное представление с

использованием серии скрытых слоев. Сеть кодировщика обучена таким образом, чтобы свести к минимуму ошибку восстановления между входными данными и восстановленными выходными данными, а также способствовать разреженности изученных функций. Это достигается за счет использования термина регуляризации, который наказывает за активацию слишком большого количества функций.

Одним из главных преимуществ разреженного автоэнкодера является то, что он способен изучать эффективные и надежные представления входных данных без необходимости в помеченных данных. Это делает его особенно полезным в ситуациях, когда помеченных данных мало или их получение дорого.

```
class SparseAutoencoder(tf.keras.Model):
    def __init__(self):
        super().__init__()
        self.encoder =  tf.keras.layers.Sequential([
             tf.keras.layers.Dense(64, activation="relu"),
             tf.keras.layers.Dense(32, activation="relu", activity_regularizer=tf.keras.regularizers.l1(0.0001)),
             tf.keras.layers.Dense(16, activation="relu", activity_regularizer=tf.keras.regularizers.l1(0.0001)),
             tf.keras.layers.Dense(8, activation="relu", activity_regularizer=tf.keras.regularizers.l1(0.0001))
        ])
        self.decoder = keras.Sequential([
             tf.keras.layers.Dense(16, activation="relu"),
             tf.keras.layers.Dense(32, activation="relu"),
             tf.keras.layers.Dense(64, activation="relu"),
             tf.keras.layers.Dense(784, activation="sigmoid"),
        ])

        self.model = tf.keras.layers.Sequential([self.encoder, self.decoder])

    def model(self):
      return self.model
```

Рисунок 16 – Код разреженног автокодировщика

9) Свёрточные нейронные сети (convolutional neural networks, CNN) и глубинные свёрточные нейронные сети (deep convolutional neural networks, DCNN) – сети, которые чаще всего применяются для обработки изображений, реже для аудио. Классификация – это наиболее частый способ применения CNN. Эти сети используют сканер», не парсящий все данные за один раз. Алгоритм обучения заключается в том, что обсчет всех пикселей не производится и проходятся по матрице ядром, который вычисляет новую

матрицу. Эти входные данные затем передаются через свёрточные слои, в которых не все узлы соединены между собой. Все эти слои сжимаются с глубиной чаще всего используются степени двойки (Рисунок 17).

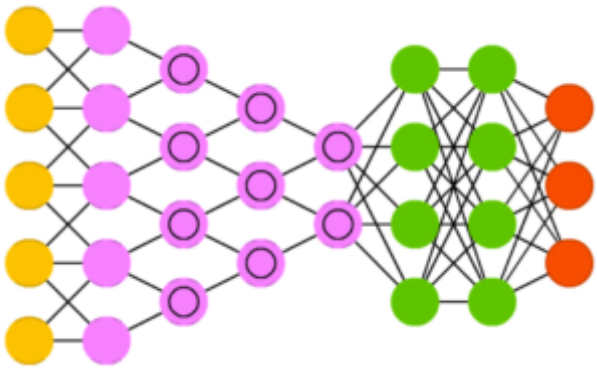


Рисунок 17 – Свёрточные нейронные сети и глубинные свёрточные нейронные сети

Сверточная нейронная сеть (CNN) – это тип модели глубокого обучения, которая особенно эффективна при обработке и анализе структурированных данных, подобных сетке, таких как изображения или последовательные данные. CNN произвели революцию в области компьютерного зрения и широко используются для различных задач, связанных с изображениями, включая классификацию изображений, обнаружение объектов и сегментацию изображений.

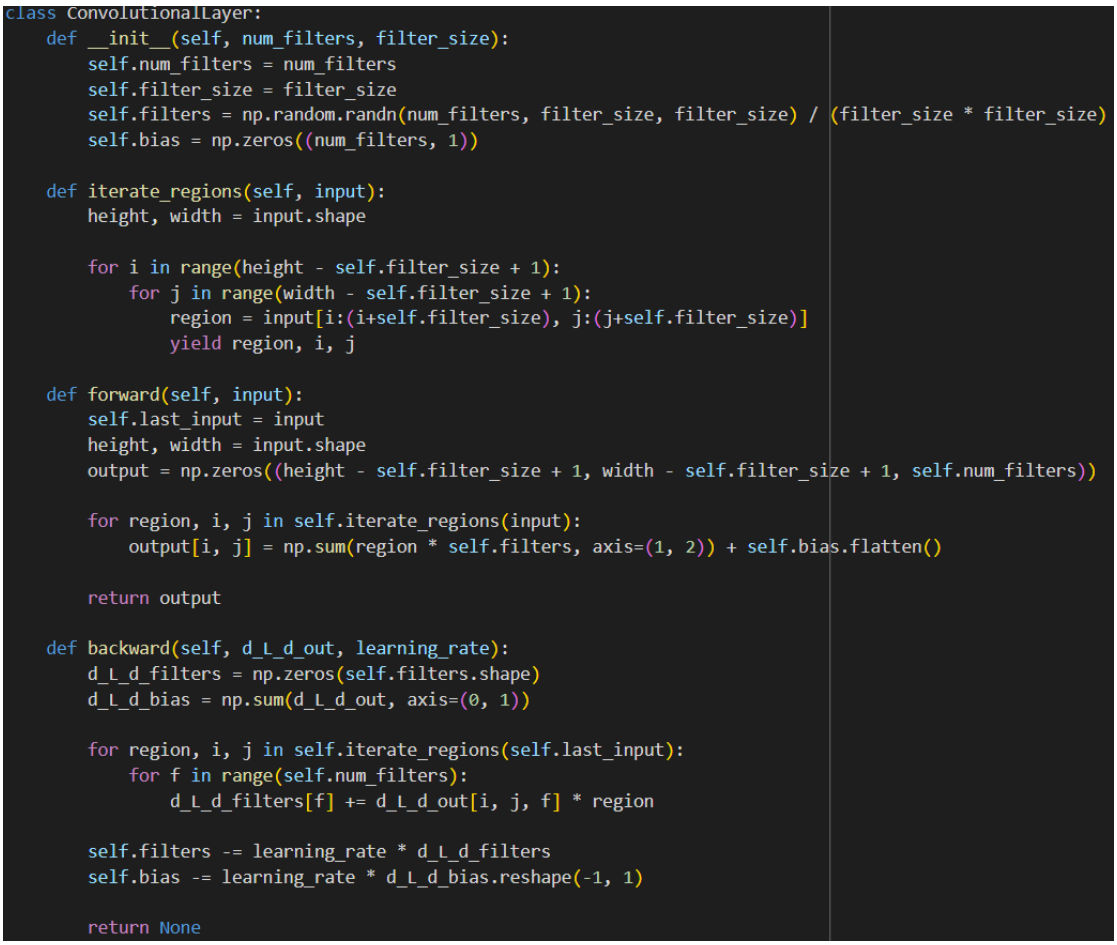
Сверточные слои являются строительными блоками CNNS. Они применяют набор обучаемых фильтров (также известных как ядра) к входному изображению, чтобы извлечь локальные объекты. Фильтры скользят по изображению и выполняют поэлементное умножение и суммирование, создавая карты объектов, которые выделяют определенные узоры или края. Сверточные слои могут изучать различные уровни абстракции - от низкоуровневых объектов, таких как ребра и углы, до объектов более высокого уровня, таких как формы и текстуры.

Объединяющие слои часто вставляются после сверточных слоев, чтобы уменьшить пространственные размеры карт объектов и уменьшить количество

параметров в сети. Наиболее распространенным типом объединения является максимальное объединение, при котором выбирается максимальное значение в пределах определенного окна или размера пула. Объединение помогает сохранить наиболее важную информацию, одновременно снижая чувствительность к небольшим пространственным изменениям и повышая эффективность вычислений.

Функции активации привносят нелинейность в CNN, позволяя ему изучать сложные взаимосвязи между входными данными и целевым объектом. Наиболее часто используемой функцией активации в CNNs является выпрямленный линейный блок (ReLU), который устанавливает отрицательные значения равными нулю и сохраняет положительные значения неизменными. ReLU помогает сети быстрее обучаться и позволяет избежать проблемы с исчезающим градиентом.

Полностью подключенные слои соединяют каждый нейрон предыдущего слоя с каждым нейроном последующего слоя, аналогично традиционным нейронным сетям. Эти слои обычно добавляются в конце сети и отвечают за составление прогнозов или классификаций на основе извлеченных объектов. Выходные данные последнего полностью подключенного уровня обычно передаются через функцию активации softmax для получения распределения вероятностей по возможным классам.



```
class ConvolutionalLayer:
    def __init__(self, num_filters, filter_size):
        self.num_filters = num_filters
        self.filter_size = filter_size
        self.filters = np.random.randn(num_filters, filter_size, filter_size) / (filter_size * filter_size)
        self.bias = np.zeros((num_filters, 1))

    def iterate_regions(self, input):
        height, width = input.shape

        for i in range(height - self.filter_size + 1):
            for j in range(width - self.filter_size + 1):
                region = input[i:(i+self.filter_size), j:(j+self.filter_size)]
                yield region, i, j

    def forward(self, input):
        self.last_input = input
        height, width = input.shape
        output = np.zeros((height - self.filter_size + 1, width - self.filter_size + 1, self.num_filters))

        for region, i, j in self.iterate_regions(input):
            output[i, j] = np.sum(region * self.filters, axis=(1, 2)) + self.bias.flatten()

        return output

    def backward(self, d_L_d_out, learning_rate):
        d_L_d_filters = np.zeros(self.filters.shape)
        d_L_d_bias = np.sum(d_L_d_out, axis=(0, 1))

        for region, i, j in self.iterate_regions(self.last_input):
            for f in range(self.num_filters):
                d_L_d_filters[f] += d_L_d_out[i, j, f] * region

        self.filters -= learning_rate * d_L_d_filters
        self.bias -= learning_rate * d_L_d_bias.reshape(-1, 1)

        return None
```

Рисунок 18 – Код свёрточного слоя

Вернемся к нейронным сетям. Её архитектура состоит из совокупности несложных процессов, разделенных на слои, в которых происходит параллельные вычисления, что позволяет ей решать задачи практически как человек. Основным отличием искусственной нейронной сети от классических алгоритмов заключается в их способности обучаться, потому что у всех нейронов есть собственный весовой коэффициент, который определяет его значимость в слое для остальных нейронов. Значимость таких программ заключается в том, что они могут принимать разумные решения с ограниченным участием человека. Поэтому сейчас к ним обращено столько внимания и тратятся огромные ресурсы

2.2 YOLO

YOLO (You Only Look Once) – это одна из самых популярных и эффективных нейронных сетей для обнаружения объектов в реальном времени. Основная идея YOLO заключается в том, что изображение делится на сетку ячеек, и каждая ячейка предсказывает несколько ограничивающих рамок и вероятности классов для каждой рамки. Затем выбираются только те рамки, которые имеют высокую вероятность содержать объект. Этот подход позволяет YOLO обрабатывать изображения очень быстро, поскольку весь процесс обнаружения выполняется за один проход через сеть.

YOLO состоит из двух основных компонентов: сверточной сети и полносвязного слоя. Сверточная сеть отвечает за извлечение признаков из изображения и преобразование его в вектор признаков. Полносвязный слой принимает этот вектор и преобразует его в тензор, содержащий информацию о рамках и классах для каждой ячейки сетки. Структура сверточной сети может быть различной, но обычно используется архитектура ResNet или Darknet. Полносвязный слой имеет форму S x S x (B x 5 + C), где S - размер сетки, B - количество рамок на ячейку, 5 - количество параметров на рамку (координаты центра, ширина, высота и вероятность объекта), C - количество классов. YOLO имеет несколько преимуществ перед другими методами обнаружения объектов. Во-первых, он очень быстрый и может работать в режиме реального времени на видеопотоках. Во-вторых, он учитывает глобальный контекст изображения и может лучше локализовать объекты разных размеров и форм. В-третьих, он менее подвержен ошибкам ложных срабатываний, поскольку он предсказывает только одну рамку на объект. Однако YOLO также имеет некоторые недостатки. Например, он может пропускать мелкие объекты или объекты, которые перекрываются друг другом. Кроме того, он может быть нестабилен при обучении и требовать тонкой настройки гиперпараметров.

Архитектура YOLO состоит из нескольких ключевых компонентов:

1) Входное изображение: входным сигналом для сети YOLO является изображение фиксированного размера.

2) Магистральная сеть: Магистральная сеть обычно представляет собой предварительно обученный deep CNN, такой как DarkNet или ResNet, который извлекает высокоуровневые функции из входного изображения. Эти функции охватывают различные уровни абстракции и помогают обнаруживать объекты различных размеров и форм.

3) Пирамида объектов: YOLO использует пирамиду объектов для извлечения объектов в различных масштабах. Эта пирамида строится путем добавления сверточных слоев с меньшими шагами к магистральной сети. Функциональная пирамида позволяет сети обнаруживать объекты разного размера и повышает производительность обнаружения.

4) Ячейки сетки: Входное изображение делится на сетку ячеек. Каждая ячейка отвечает за обнаружение объектов, которые попадают в ее пространственное местоположение.

5) Якорные блоки: YOLO использует якорные блоки для обнаружения объектов различной формы и соотношения сторон. Рамки привязки – это заранее определенные ограничивающие рамки различных размеров и соотношений сторон, которые размещаются в каждой ячейке сетки. Сеть предсказывает координаты ограничивающего прямоугольника относительно опорных прямоугольников.

6) Оценка объектности: Каждая ячейка сетки предсказывает оценку объектности, которая представляет вероятность наличия объекта. Этот показатель помогает отфильтровать фоновые области и сосредоточиться на регионах с высокой вероятностью обнаружения объектов.

7) Предсказания классов: YOLO предсказывает вероятности классов для каждой ячейки сетки. Количество классов обычно предопределено в зависимости от домена приложения. Сеть выводит распределение вероятностей для каждого класса, указывающее вероятность принадлежности объекта к определенному классу.

8) Немаксимальное подавление (NMS): после генерации ограничивающих рамок и соответствующих им вероятностей классов YOLO применяет немаксимальное подавление, чтобы удалить повторяющиеся обнаружения и выбрать наиболее точные ограничивающие рамки. NMS гарантирует, что каждый объект обнаруживается только один раз, и устраняет избыточные ограничивающие рамки.

9) Выходные данные: Окончательный вывод YOLO состоит из координат ограничивающего прямоугольника, оценок объектности и вероятностей классов для обнаруженных объектов на входном изображении.

10) Со временем YOLO эволюционировал, выпустив несколько версий, таких как YOLOv1, YOLOv2 (также известный как YOLO9000), YOLOv3 и YOLOv4. Каждая версия содержит архитектурные улучшения и оптимизацию для повышения точности и скорости обнаружения.

Архитектура YOLO, обладающая способностью обрабатывать изображения в режиме реального времени при сохранении хорошей производительности обнаружения, нашла применение в различных областях, включая автономное вождение, наблюдение и робототехнику.

3 Создание web-приложения

3.1 Выбор тематики

После выбора фреймворков нужно определиться с необходимым функционалом. Нужно создать web-приложение, содержащее нейронную сеть. Самый простой пример – сайт для распознавания кошек и собак на изображении.

Функционал:

1) Скачивание изображения со стороны клиента, т.е. на фронтенд части приложения.
2) Принятие изображения на стороне сервера.
3) Отправка в нейронную сеть для анализа, часть сервера это и есть сеть.
4) Отправка обработанных нейронной сетью данных на сервер.
5) Отправка данных на фронтенд часть сервера.

3.2 Реализация web-приложения

3.2.1 Фронтенд

Создание всех проектов было рассказано выше. Повторим все шаблоны по созданию.

Начнем с фроненд части сервера (Рисунок 19).

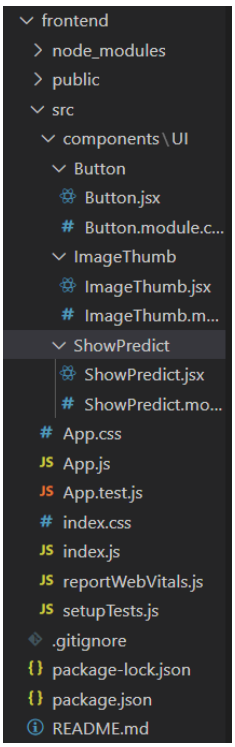


Рисунок 19 – Архитектура фронтенд части приложения

В главном компоненте «App» вызываются компоненты, объявлены функции для работы с данными, импорты со стилями, axios – библиотека для работы с запросами, useState – библиотека для работы с переменными и состояниями (Рисунок 20).

В нем реализован весь функционал сайта:

1) Загрузка и отображение изображения (функции imgUpload и ImageThumb) хранящиеся на фронтенд части приложения;

2) Отправка изображения на сервер (функция uploadImg);

3) Получение и отображение (функция ShowPredict) обработанных данных с сервера;

4) Отображение изображения, пришедший от сервера.

```
import React, { useState } from "react";
import axios from 'axios';
import ImageThumb from "./components/UI/ImageThumb/ImageThumb";
import './App.css';
import Button from "./components/UI/Button/Button";
import ShowLabel from "./components/UI/ShowLabel/ShowLabel";

function App() {

  const [file, setFile] = useState(null)
  const [label, setLabel] = useState(null)

  const imgUpload = e => {
    setFile(e.target.files[0]);
  }

function dataURLtoFile(dataurl) {

  let arr = dataurl.split(','),
      mime = arr[0].match(/:(.*?);/)[1],
      bstr = atob(arr[1]),
      n = bstr.length,
      u8arr = new Uint8Array(n);

  while(n--){
      u8arr[n] = bstr.charCodeAt(n);
  }

  return new File([u8arr], {type:mime});
}

  const uploadImg = async (e) =>{
    e.preventDefault();
    const formData = new FormData();
      formData.append(
        "file",
        file,
        file.name
      );
    const img = await axios.post('http://localhost:8000/fish/', formData).then(data => {
      setLabel(data.data.label)
        console.log(data.data)
      const base64 = `${data.data.path_img}`

      const file = dataURLtoFile(base64);

      const url = URL.createObjectURL(file);
      document.querySelector('img').src = url;
    });
  };
    return(
      <div className="App">
      <header className="App-header">
      <form onSubmit={(e)=>uploadImg(e)}>
        <ShowLabel label = {label}/>
        <ImageThumb image={file} />
        <img/>
        <div className="file-upload">
        <label>
          <input onChange = {imgUpload} name = 'image' type = 'file' accept=".jpeg, .png, .jpg"/>
          <span>Choose file</span>
        </label>
        </div>
        <Button />
      </form>
      </header>
    </div>
    );
}

export default App;
```

Рисунок 20 – Компонент App

В компоненте ShowPredict (Рисунок 21) реализован вывод вероятности вида животного, выданной нейронной сетью.

```
import React from 'react'

export default function ShowPredict( {pred} ) {
  return (
    <div className = 'ShowRow'>
      {pred}
    </div>
  )
}
```

Рисунок 21 – Компонент ShowPredict

Компонент ImageThumb (Рисунок 22) принимает и отображает изображение на лицевой части сайта.

```
import React from "react";
import classes from './ImageThumb.module.css'

export default function ImageThumb({ image }) {
    return(
        <div className = {classes.ImgBox}>
            {image && <img src={URL.createObjectURL(image)} alt={image.name} />}
        </div>
        )
};
```

Рисунок 22 – Компонент ImageThumb

Компонент Button выполняет функцию кнопки (Рисунок 23).

```
import React from 'react'
import classes from './Button.module.css'

export default function Button() {
  return (
    <div>
        <button type="submit" className={classes.Button}>Upload</button>
    </div>
  )
}
```

Рисунок 23 – Компонент Button

3.2.2 Бэкенд

Бэкенд часть web-приложения должна выполнять следующие действия:

1) Принимать данные с фронтенд части.
2) Оправлять нейронной сети для прогноза.
3) Принимать прогноз от сети.
4) Отправлять на фронтенд части.

Все эти функции реализованы и представлены на

Рисунок 24.

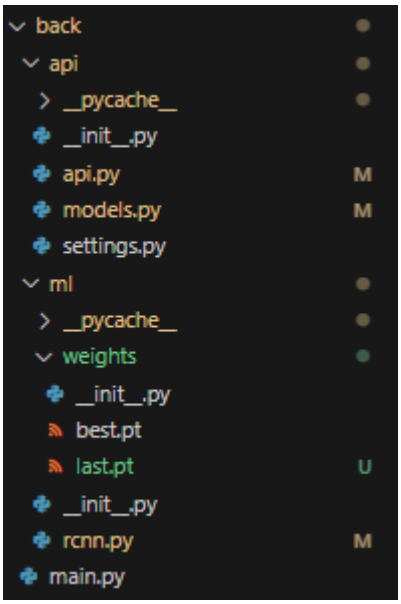


Рисунок 24 – Архитектура бэкенд части

В файле api.py реализовано соединение с фронтенд частью приложения.
В функции create_upload_file реализовано получение картинки, передача её нейронной сети и получение предикта. Функция to_user отправляет предикт, полученный от нейронной сети (

Рисунок 25).

```python
from fastapi import File, UploadFile
from .models import *
import base64
from .settings import app
from ml.rcnn import predict


@app.post("/fish/")
async def create_upload_file(file: UploadFile = File(...)):
    img = await file.read()
    pred = predict(img)
    with open(pred['path_img'], 'rb') as f:
        base64image = base64.b64encode(f.read())
    pred['path_img'] = base64image

    return pred
```

Рисунок 25 – Файл app.py бэкенд части

В файле settings.py реализован CORS policy. В нем перечислены доступные правила для HTTP запросов.

```python
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware

app = FastAPI()

origins = ['http://localhost:3000']

app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

Рисунок 26 – Файл settings.py CORS policy

В файле ml.py реализованы функции:

1) prepare_img, которая подгоняет картинку под необходимый размер и переводит в необходимый формат для нейронной сети.

2) predict хранит модель, вызывает её, отправляет изображение с детектированным животным, отправляет вид животного и вероятность отнесения к данному виду.

```python
from ultralytics import YOLO
from PIL import Image
from io import BytesIO
from api.models import Predict
import os


CLASSES = {0: 'Sprat', 1: 'Golden_crucian', 2: 'Gold_fish', 3: 'Pangas'}
model = YOLO(r'.\ml\weights\last.pt')


def prepare_img(path:str) -> Image.Image:
    image = Image.open(BytesIO(path))
    newsize = (640, 640)
    image = image.resize(newsize)

    return image


def predict(path:str) -> Predict:
    img = prepare_img(path)
    model.predict(source=img, save_txt=True, save=True)

    obsolete_path = "../runs/detect/"
    files = os.listdir(obsolete_path)
    with open(f'{obsolete_path}{files[-1]}/labels/image0.txt', 'r', encoding='utf-8') as f:
        pred = f.read()

    pred = pred.split()

    path_img = f'{obsolete_path}{files[-1]}/image0.jpg'

    label = CLASSES[int(pred[0])]

    return {'label': label, 'path_img': path_img}
```

Рисунок 27 – Файл ml.py

Файл main.py содержит точку запуска всего web-приложения (Рисунок 28).

```python
import uvicorn


if __name__ == "__main__":
    uvicorn.run("app.api:app", host="0.0.0.0", port=8000, reload=True)
```

Рисунок 28 – Файл main.py

3.3 Результат работы

Разработанное web-приложение без введенных данных представлено на Рисунок 29.

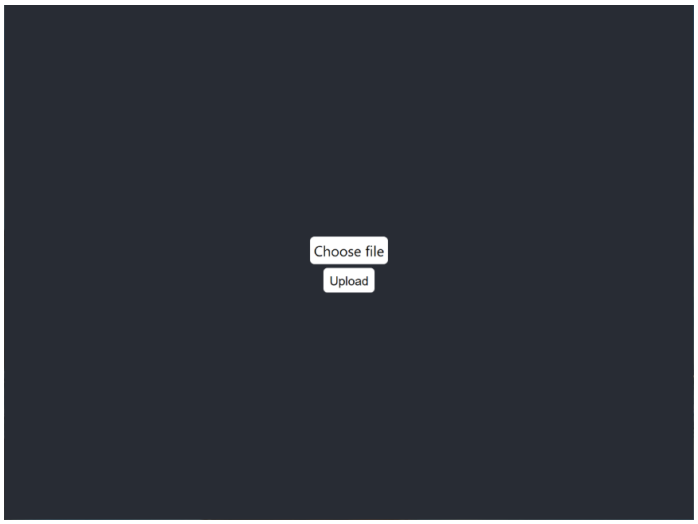


Рисунок 29 – Первоначальный вид сайта

После отправки изображения, сайт сначала отображает картинку (Рисунок 30).

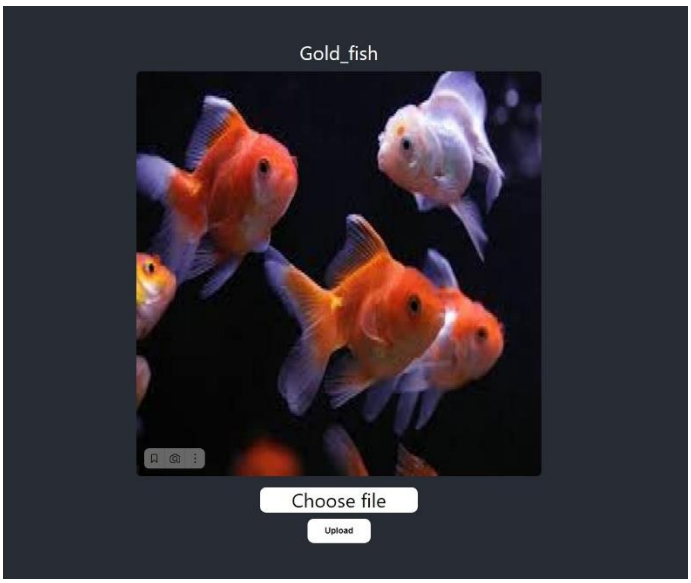


Рисунок 30 – После ввода изображения

При нажатии на кнопку «Upload» файл отправляется на сервер, там обрабатывается и отображается результат (Рисунок 31).

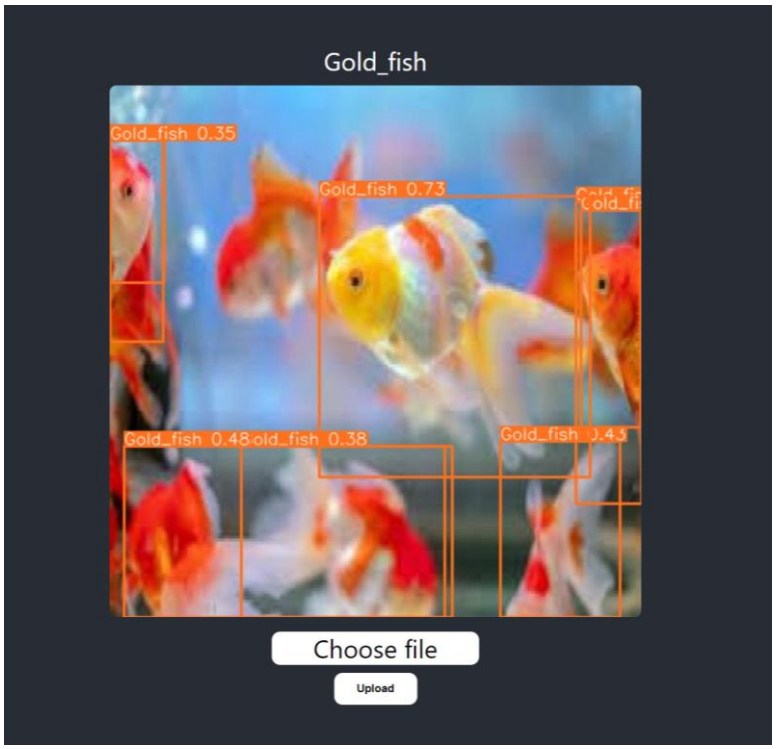


Рисунок 31 – После вывода результата

ЗАКЛЮЧЕНИЕ

В результате нами была достигнута поставленная цель – разработать web-приложение по детекции рыбок на изображении (нейронной сетью).

Функционал сайта заключается в том, что пользователь отправляет картинку и сайт детектирует рыбок на изображении.

Актуальность нашей работы заключается в популярности нейросетевых технологий, особенно в web-приложениях, так как они пользуются высокой популярностью. Роль моделей машинного обучения заключается в автоматизации работы и обработке огромного количества данных.

Для реализации поставленной цели нужно было решить следующие задачи:

- изучить основы разработки web-приложений;
- изучить основы разработки web-приложений;
- ознакомится с подходящими инструментами для разработки web-приложений;
- выбрать датасет и архитектуру сети,
- изучить основы React.js, FastAPI, Object detection;
- создать полноценное web-приложение.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Web-приложение: понятие, компоненты и принципы работы: сайт – 2019. – URL: https://smartiqa.ru/courses/web/lesson-1 (дата обращения 14.05.2023).

2. Общие сведения о веб-приложениях: сайт – 2010. – URL: https://helpx.adobe.com/ru/dreamweaver/using/web-applications.html (дата обращения 16.05.2023).

3. Как работают веб-приложения: сайт – 2015. – URL: https://habr.com/ru/post/450282/ (дата обращения 18.05.2023).

4. Для чего строят и обучают нейросети в IT: сайт – 2017. – URL: https://practicum.yandex.ru/blog/chto-takoe-neyronnye-seti/ (дата обращения 16.12.2022).

5. В каких случаях стоит использовать Django (а в каких не стоит): сайт – 2015. – URL: https://habr.com/ru/company/piter/blog/449784/ (21.05.2023).

6. FastAPI vs Flask: сайт – 2014. – URL: https://christophergs.com/python /2021/06/16/python-flask-fastapi/ (дата обращения 22.05.2023).

7. FastAPI сайт – 2019. – URL: https://fastapi.tiangolo.com/ (дата обращения 24.05.2023).

8. Шпаргалка по разновидностям нейронных сетей. Часть первая. Элементарные конфигурации: сайт – 2017. – URL: https://tproger.ru/translations/neural-network-zoo-1/ (дата обращения 24.05.2023).

9. YOLO: You Only Look Once.: сайт – 2021. – URL: https://medium.com/analytics-vidhya/yolo-you-look-only-once-9af63cb143b7 (дата обращения 20.05.2023).