**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

Специальность 1-40 05 01-01 Информационные системы и технологии (в проектировании и производстве)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

по дисциплине «Оптимизация проектных решений»

на тему: «**ПРОГРАММНОЕ СРЕДСТВО ДЛЯ РЕШЕНИЯ ЗАДАЧИ ОБНАРУЖЕНИЯ ОГУРЦОВ НА ОСНОВЕ *YOLOv5***»

Исполнитель: студент гр. ИТП-41

Расшивалов Н.И.

Руководитель: доцент

Мурашко И.А.

Дата проверки: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата допуска к защите: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подписи членов комиссии

по защите курсового проекта: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Гомель 2022

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc120832481)

[1 СРЕДСТВА ДЛЯ РЕАЛИЗАЦИИ НЕЙРОННЫХ СЕТЕЙ 5](#_Toc120832482)

[1.1 Задача обнаружения объектов на изображении 5](#_Toc120832483)

[1.2 Оптимизаторы в нейронных сетях 10](#_Toc120832484)

[1.3 Средства разработки серверного программного обеспечения 11](#_Toc120832485)

[1.4 Средства разработки клиентского программного обеспечения 13](#_Toc120832486)

[2 АРХИТЕКТУРА ПРОГРАММНОГО КОМПЛЕКСА ПО ОБНАРУЖЕНИЮ ОГУРЦОВ 15](#_Toc120832487)

[2.1 Архитектура клиент-серверного приложения по обнаружению   
огурцов 15](#_Toc120832488)

[2.2Топологии нейронных сетей 16](#_Toc120832489)

[2.3 Архитектура сверточных нейронных сетей 17](#_Toc120832490)

[2.4 Сравнение фреймворков глубокого обучения 19](#_Toc120832491)

[2.5 Архитектура *YOLOv5* 20](#_Toc120832492)

[3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ 23](#_Toc120832493)

[3.1 Структура разработанного программного комплекса 23](#_Toc120832494)

[3.2 Набор данных 25](#_Toc120832495)

[3.3 Обучение модели *YOLOv5* 26](#_Toc120832496)

[3.4 Функционал разработанного приложения 27](#_Toc120832497)

[ЗАКЛЮЧЕНИЕ 29](#_Toc120832498)

[Список использованных источников 30](#_Toc120832499)

[ПРИЛОЖЕНИЕ А Листинг программы 31](#_Toc120832500)

[ПРИЛОЖЕНИЕ Б Схема алгоритма 38](#_Toc120832501)

# **ВВЕДЕНИЕ**

В последнее время все больше и больше людей говорят о так называемых нейронных сетях, которые в скором времени, по оценкам и прогнозам специалистов, будут уверенно применяться в робототехнике, в машиностроении и в других совершенно разных сферах человеческой деятельности. Что это за нейронные сети, как они работают, как они используются и как они могут стать полезными для нас?

Нейронные сети являются одним из направлений научных исследований в области создания искусственного интеллекта, в основе которого лежит стремление подражать нервной системе человека. Нейронные сети основаны на примитивной биологической модели нервной системы.

Преимущества нейронных сетей, обусловлены возможностью распараллеливания обработки информации, а также самообучением, т. е. возможностью обобщать – способность получать обоснованный результат на основании данных, не встречавшихся ранее в процессе обучения. Указанные преимущества позволяют искусственным нейронным сетям решать сложные задачи, считающиеся на сегодняшний день трудноразрешимыми.

Актуальность разработки данного программного обеспечения обусловлена необходимостью автоматизирования различных процессов, которые невозможны без использования нейронных сетей.

Целью курсового проекта является разработка программного обеспечения, позволяющего производить определение объектов на заданных изображениях с использованием искусственных нейронных сетей.

Задачи курсового проекта можно определить следующим перечнем:

– подготовить набор данных;

– обучить нейронную сеть;

– разработать программное обеспечение, позволяющее использовать обученную нейронную сеть.

# **1 СРЕДСТВА ДЛЯ РЕАЛИЗАЦИИ НЕЙРОННЫХ СЕТЕЙ**

* 1. **Задача обнаружения объектов на изображении**

Обнаружение объектов – это метод [компьютерного зрения](https://heartbeat.fritz.ai/the-5-computer-vision-techniques-that-will-change-how-you-see-the-world-1ee19334354b), целью которого является обнаружение таких объектов, как автомобили, здания и люди, и это лишь некоторые из них. Обычно объекты можно идентифицировать [либо по изображениям, либо по видеоканалам](https://heartbeat.fritz.ai/detecting-objects-in-videos-and-camera-feeds-using-keras-opencv-and-imageai-c869fe1ebcdb).

[Обнаружение объектов](https://www.fritz.ai/features/object-detection.html) широко применяется в видеонаблюдении, беспилотных автомобилях и [отслеживании объектов/людей](https://heartbeat.fritz.ai/real-time-person-tracking-on-the-edge-with-a-raspberry-pi-93ae636af9fa). В этой главе рассматриваются основы обнаружения объектов и некоторые из наиболее часто используемых алгоритмов, а также несколько совершенно новых подходов.

Обнаружение объекта идентифицирует присутствие объекта на изображении и рисует ограничивающую рамку вокруг этого объекта. Обычно это включает два процесса:

* классификацию и тип объекта;
* рисование рамки вокруг этого объекта.

Одни из самых распространенных архитектур моделей, используемых для обнаружения объектов [1, c-504]:

* *R-CNN*;
* *Fast R-CNN*;
* *Faster R-CNN*;
* *Mask R-CNN*;
* *YOLO*.

***1.1.1*** Модель*R-CNN* сочетает в себе два основных подхода:

* применение высокопроизводительных сверточных нейронных сетей к восходящим предложениям регионов для локализации и сегментации объектов;
* контролируемая предварительная подготовка к вспомогательным задачам.

Архитектура сети *R-CNN (Regions With CNNs*) была разработана командой из *UC* *Berkley* для применения *Convolution* *Neural* *Networks* к задаче *object* *detection*. Существовавшие на тот момент подходы к решению таких задач приблизились к максимуму своих возможностей и значимо улучшить их показатели не получалось.

*CNN* хорошо показывали себя в классификации изображений, и в данной сети они по сути были применены для того же самого [2, c-202]. Для этого на вход *CNN* подавалось не всё изображение целиком, а предварительно выделенные другим способом регионы, на которых предположительно имеются какие-то объекты.

В качестве *CNN*-сети использовалась так же готовая архитектура – *CaffeNet* (*AlexNet*). Такие нейросети, как и другие для набора изображений *ImageNet*, проводят классификацию на 1000 классов. *R-CNN* разрабатывалась для детектирования объектов меньшего количества классов (*N*= 20 или 200), поэтому последний классификационный слой *CaffeNet* был заменён на слой с *N*+1 выходами (с дополнительным классом для фона).

Несмотря на то, что *CNN* тренировалась на распознавание *N*+1 классов, в итоге она использовалась только для извлечения фиксированного 4096-размерного вектора признаков. Непосредственным определением объекта на изображении занимались *N* линейных *SVM*, каждый из которых проводил бинарную классификацию по своему типу объектов, определяя есть ли такой в переданном регионе или нет. В оригинальном документе вся процедура иллюстрируется схемой, которая указана на рисунке 1.1.

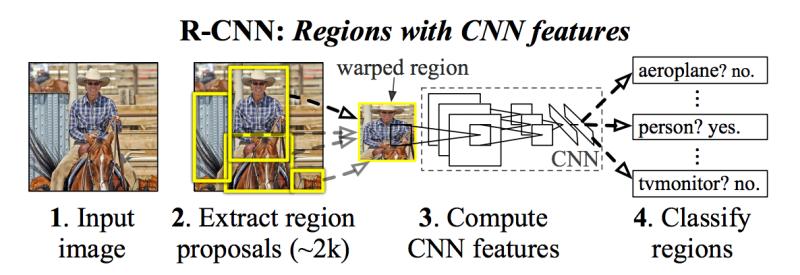


Рисунок 1.1 – Процедура извлечения признаков *CNN*

***1.1.2*** Несмотря на высокие результаты, производительность *R-CNN* была всё же невысока, особенно для более глубоких, чем *CaffeNet* сетей (таких как *VGG16*). Кроме того, обучение *bounding* *box* *regressor* и *SVM* требовало сохранения на диск большого количества признаков, поэтому оно было дорогим с точки зрения размера хранилища.

Авторы *Fast* *R-CNN* предложили ускорить процесс за счёт пары модификаций. Пропускать через *CNN* не каждый из 2000 регионов-кандидатов по отдельности, а всё изображение целиком. Предложенные регионы потом накладываются на полученную общую карту признаков.

Вместо независимого обучения трёх моделей (*CNN*, *SVM*, *bbox* *regressor*) совместить все процедуры тренировки в одну.

Преобразование признаков, попавших в разные регионы, к фиксированному размеру производилось при помощи процедуры *RoIPooling*. Окно региона шириной *w* и высотой *h* делилось на сетку, имеющую *H×W* ячеек размером *h/H × w/W*. По каждой такой ячейке проводился *Max* *Pooling* для выбора только одного значения, давая таким образом результирующую матрицу признаков *H×W*.

Бинарные *SVM* не использовались, вместо этого выбранные признаки передавались на полносвязанный слой, а затем на два параллельных слоя: *softmax* с *K+1* выходами (по одному на каждый класс + 1 для фона) и *bounding* *box* *regressor*. Общая архитектура сети выглядит изображена на рисунке 1.2.

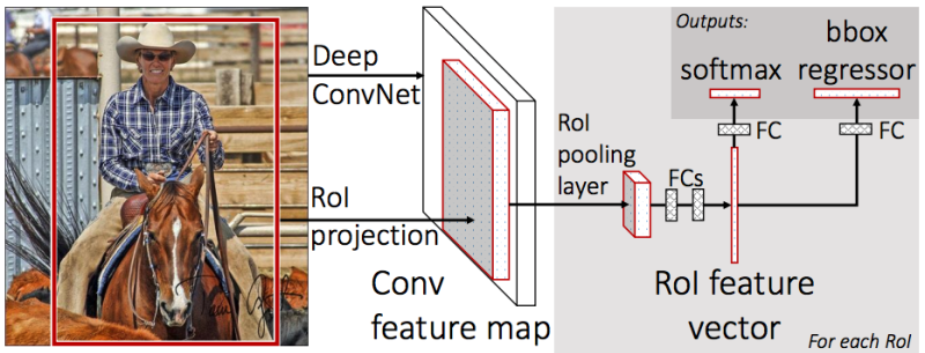


Рисунок 1.2 – Общая архитектура *Fast* *R-CNN*

***1.1.3*** *Faster R-CNN.* После улучшений, сделанных в *Fast R-CNN*, самым узким местом нейросети оказался механизм генерации регионов-кандидатов. В 2015 команда из *Microsoft* *Research* смогла сделать этот этап значительно более быстрым. Они предложили вычислять регионы не по изначальному изображению, а опять же по карте признаков, полученных из CNN. Для этого был добавлен модуль под названием *Region* *Proposal* *Network* (*RPN*). Новая архитектура целиком представлена на рисунке 1.3.

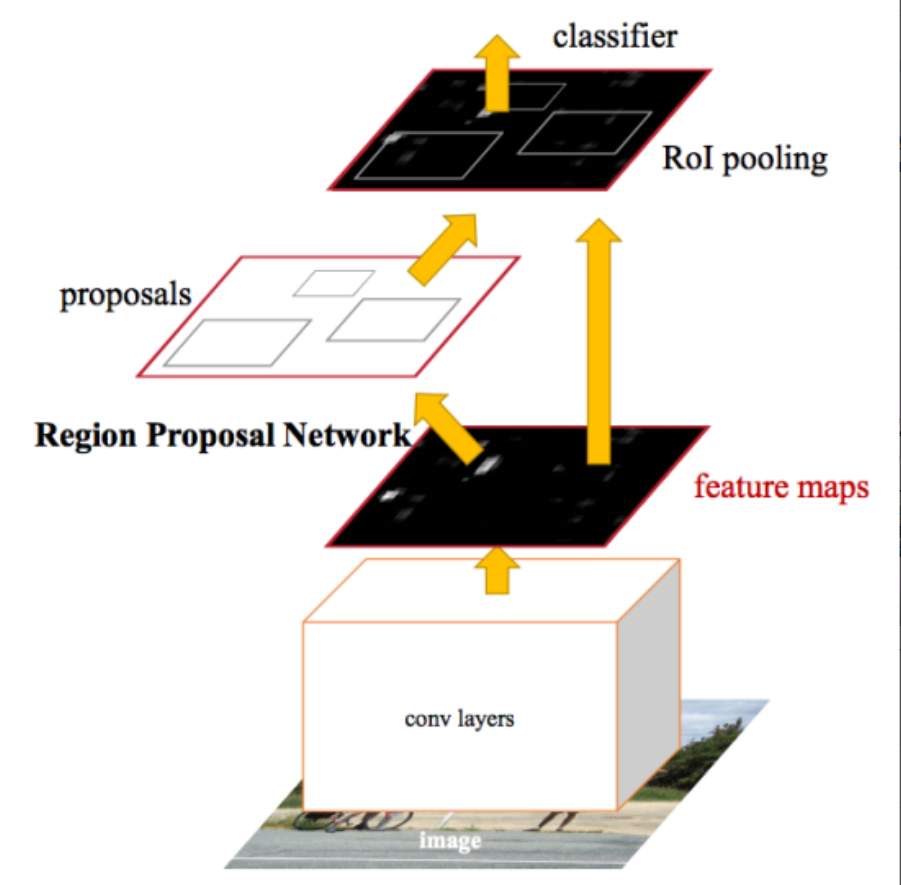


Рисунок 1.3 – Общая архитектура *Faster* *R-CNN*

В рамках *RPN* по извлечённым *CNN* признакам скользят «мини-нейросетью» с небольшим (3*х*3) окном. Полученные с её помощью значения передаются в два параллельных полносвязанных слоя: *box-regression layer (reg)* и *box-classification layer (cls).* Выходы этих слоёв базируются на так называемых *anchor*-ах – *k* рамках для каждого положения скользящего окна, имеющих разные размеры и соотношения сторон. *Reg*-слой для каждого такого *anchor*-а выдаёт по четыре координаты, корректирующие положение охватывающей рамки, *cls*-слой выдаёт по два числа – вероятности того, что рамка содержит хоть какой-то объект или что не содержит.

Для того, чтобы разделять признаки, получаемые в *CNN*, между *RPN* и модулем детектирования, процесс обучения всей сети построен итерационно, с использованием нескольких шагов:

* инициализируется и обучается на определение регионов-кандидатов *RPN*-часть;
* с использованием предлагаемых *RPN* регионов заново обучается *Fast* *R-CNN* часть;
* Обученная сеть детектирования используется, чтобы инициализировать веса для *RPN*. Общие *convolution*-слои, однако, фиксируются и производится донастройка только слоёв, специфичных для *RPN*.
* С зафиксированными *convolution*-слоями окончательно донастраивается *Fast R-CNN.*

***1.1.4***Алгоритм *YOLO* (*You Look Only Once*), предложенный в 2016 году, был первой попыткой сделать возможной детекцию объектов в реальном времени. В рамках алгоритма *YOLO* исходное изображение сначала разбивается на сетку из ячеек. Если центр объекта попадает внутрь координат ячейки, то эта ячейка считается ответственной за определение параметров местонахождения объекта. Каждая ячейка описывает несколько вариантов местоположения ограничивающих рамок для одного и того же объекта. Каждый из этих вариантов характеризуется пятью значениями – координатами центра ограничивающей рамки, его шириной и высотой, а также степени уверенности в том, что ограничивающая рамка содержит в себе объект. Также необходимо для каждой пары класса объектов и ячейки определить вероятность того, что ячейка содержит в себе объект этого класса.

Алгоритм *YOLO* работает быстрее алгоритмов семейства *R-CNN* за счёт того, что поддерживает дробление на константное количество ячеек вместо того, чтобы предлагать регионы и рассчитывать решение для каждого региона отдельно, однако, в качестве проблем *YOLO* указывается плохое качество распознавания объектов сложной формы или группы небольших объектов из-за ограниченного числа кандидатов для ограничивающих рамок [3].

Принцип работы *YOLO* представлен на рисунке 1.4.

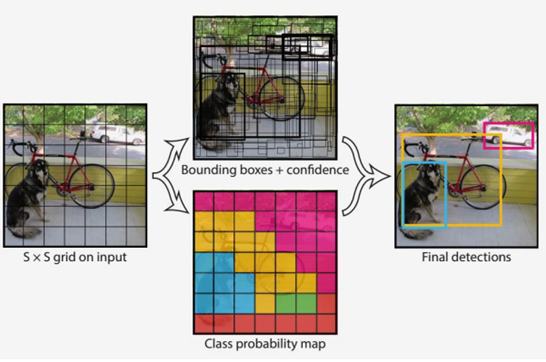


Рисунок 1.4 – Принцип работы *YOLO*

В качестве алгоритма нейронной сети выбран алгоритм *YOLO* так он имеет существенные преимущества перед *R-CNN* и *Fast R-CNN*, учитывая, что данная задача не требует работы в реальном времени, алгоритм *YOLO* позволяет более точно обнаруживать объекты в сравнении с *Fast R-CNN*.

## **1.2 Оптимизаторы в нейронных сетях**

Оптимизаторы – это алгоритмы или методы, используемые для минимизации функции ошибок (функции потерь) или для максимизации эффективности обучения. Оптимизаторы – это математические функции, которые зависят от обучаемых параметров модели, то есть весов и смещений. Оптимизаторы помогают узнать, как изменить веса и скорость обучения нейронной сети, чтобы уменьшить потери.

***1.2.1*** Градиентный спуск – это алгоритм оптимизации, основанный на выпуклой функции и итеративно изменяющий ее параметры, чтобы минимизировать заданную функцию до ее локального минимума. Градиентный спуск итеративно уменьшает функцию потерь, двигаясь в направлении, противоположном направлению самого крутого подъема. Это зависит от производных функции потерь для нахождения минимумов. использует данные всего обучающего набора для вычисления градиента функции затрат к параметрам, что требует большого объема памяти и замедляет процесс.

***1.2.2*** Градиентный спуск с импульсом. Это метод стохастической оптимизации, который добавляет понятие импульса к обычному стохастическому градиентному спуску. Импульс имитирует инерцию объекта, когда он движется, то есть направление предыдущего обновления сохраняется в определенной степени во время обновления, в то время как текущий градиент обновления используется для точной настройки окончательного направления обновления. Таким образом, можно в определенной степени повысить стабильность, чтобы быстрее учиться, а также иметь возможность избавиться от локальной оптимизации.

***1.2.3*** *AdaGrad* (*Adaptive Gradient Descent*). Во всех алгоритмах, обсуждаемых ранее, скорость обучения остается постоянной. Особенность, лежащая в основе *AdaGrad*, заключается в том, можно использовать разные скорости обучения для каждого нейрона для каждого скрытого слоя на основе разных итераций.

Преимущества *AdaGrad*:

– скорость обучения меняется адаптивно с итерациями;

– способен обучать разреженные данные.

Недостатки *AdaGrad*:

– если нейронная сеть глубока, скорость обучения становится очень ма-лой.

***1.2.4*** *Adadelta* – это расширение *Adagrad*, которое пытается уменьшить агрессивность *Adagrad*, монотонно снижая скорость обучения, и устранить проблему замедления скорости обучения. В *Adadelta* нет необходимости устанавливать скорость обучения по умолчанию, поскольку берется отношение среднего значения предыдущих временных шагов к текущему градиенту.

Преимущества *Adadelta*:

– не нужно устанавливать скорость обучения по умолчанию.

Недостатки *Adadelta*:

– повышенные требования к вычислительной мощности.

***1.2.5*** *RMSProp* (*Root Mean Square Propagation*) – это специальная версия *Adagrad*, в которой скорость обучения представляет собой экспоненциальное среднее значение градиентов, а не совокупную сумму квадратов градиентов. *RMS-Prop* в основном сочетает в себе momentum с *AdaGrad*.

***1.2.6*** *Adam* (*Adaptive Moment Estimation*) оптимизатор – один из самых популярных и известных алгоритмов оптимизации с градиентным спуском. Это метод, который вычисляет адаптивные скорости обучения для каждого параметра. Он хранит как затухающее среднее значение прошлых градиентов, аналогичное *momentum*, так и затухающее среднее значение прошлых квадратов градиентов, аналогичное *RMS-Prop* и *Adadelta*. Таким образом, он сочетает в себе преимущества обоих методов.

## **1.3 Средства разработки серверного программного обеспечения**

*Python* и *C*# являются высокоуровневыми объектно-ориентированными языками программирования. Их легко изучать, они обеспечивают быструю разработку и отличную производительность. *Python* широко используется для веб-разработки, разработки программного обеспечения, анализа данных, визуализации данных и автоматизации задач. С другой стороны, *C*# – популярный выбор для создания веб-приложений, настольных приложений и веб-сервисов.

Свой синтаксис *C#* во многом унаследовал от *C*++ и *Java*. Разработчики, имеющие опыт написания приложений на этих языках, найдут в C# много знакомых черт. Но в то же время он в значительной степени инновационный, благодаря атрибутам, делегатам, событиям и другим возможностям, которые позволили использовать новые подходы в программировании. Принципиально важным отличием от своих предшественников является изначальная ориентация на безопасность кода за счет наличия среды *CLR*, при неуправляемом программировании программисты зачастую забывают освободить память, ставшую ненужной, или пытаются использовать уже освобожденную, что приводит к самым опасным и непредвиденным ошибкам в приложении.

Переняв многое от своих предшественников, опираясь на практику их использования, C# исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++, не поддерживает множественное наследование классов (между тем допускается множественная реализация интерфейсов).

Преимущества C#:

– *C*# – простой, надежный и масштабируемый язык программирования;

– динамически типизированный характер *C*# облегчает разработчикам поиск ошибок в коде.

– *C*# устраняет проблему утечки памяти.

Недостатки *C*#:

– существование ряда ограничений при разработке для операционных систем отличных от *Windows*;

– *C*# не такой гибкий, как другие языки программирования, так как он зависит от платформы .*NET*.

*Python* – это универсальный интерпретируемый язык высокого уровня. Стиль *Python* – значительные отступы, которые подчеркивают удобочитаемость кода. Он следует нескольким принципам программирования, таким как объектно-ориентированное, функциональное, структурированное, рефлексивное и процедурное. *Python* включает обширную стандартную библиотеку, которая значительно упрощает разработку программ.

*Python* хорошо известен своей читабельностью кода. Кроме того, *Python* легко изучить и понять, поскольку в его синтаксисе используются простые английские ключевые слова и не используются фигурные скобки для разделения блоков.

Преимущества *Python*:

– *Python* – это язык с динамической типизацией. Это означает, что нет необходимости определять тип данных переменной, поскольку она автоматически присваивает типы данных переменным во время выполнения;

– *Python* легко читать и изучать благодаря синтаксису, похожему на английский. Кроме того, исключается использование точки с запятой после конца оператора и разделителей для начала и конца блока;

– *Python* является интерпретируемым языком, он выполняет код построчно, останавливает выполнение в случае ошибки и сообщает об этом.

– *Python* совместим и переносится в системы *Windows*, *macOS* и *Unix* / *Linux*;

* Имеет широкий спектр возможностей по работе с нейронными сетями.

Недостатки *Python*:

– *Python* имеет низкую скорость, потому что это интерпретируемый язык, и он выполняет код построчно;

– Это не идеальный выбор для задач с интенсивным использованием памяти, поскольку он потребляет большой объем памяти из-за гибкости типов данных;

– Поскольку *Python* неэффективен с точки зрения памяти и имеет медленную вычислительную мощность, он не используется при разработке клиентских или мобильных приложений.

Так как *Python* предоставляет наиболее широкий спектр возможностей по реализации нейронных сетей, данный язык программирования и будет использоваться для создания вычислительного сервера.

## **1.4 Средства разработки клиентского программного обеспечения**

При разработке клиентских веб-приложений, для упрощения их создания часто используют современные фреймворки *JavaScript*, такие как *React* или *Angular.*

Основными критериями по выбору фреймворка, являются следующие параметры:

– Время разработки;

– Процесс разработки;

– Технологические возможности фреймворка.

*React* предлагает простое и функциональное создание компонентов, а также пропагандирует их использование для поддержания элегантного кода *API*. Имея большой выбор легкодоступных открытых плагинов и расширений, можно разработать практически любой тип веб-сайта.

Особенности *React*:

– компонентно-ориентированный;

– декларативный;

– производительный (благодаря *React Virtual DOM*);

– серверный рендеринг.

Недостатки:

– *React* не однозначен и оставляет разработчикам возможность выбирать лучший способ развития. Это может быть решено сильным лидерством проекта и хорошими процессами.

Особенности *Angular*:

– универсальный;

– нативный;

– используется с *Typescript*;

– интеллектуальное автозаполнение *HTML*-компонента;

– архитектура рассчитана на крупные проекты;

– *MVVM*-модель.

Недостатки:

– достаточно сложный для изучения;

– слабая производительность.

Исходя из рассмотренных фреймворков выбран *React* который идеально подходит для небольших проектов и имеет хорошую производительность.

# **2 АРХИТЕКТУРА ПРОГРАММНОГО КОМПЛЕКСА ПО ОБНАРУЖЕНИЮ ОГУРЦОВ**

## **2.1 Архитектура клиент-серверного приложения по обнаружению огурцов**

Программное обеспечение использует архитектуру «Клиент-Сервер» которая предусматривает разделение процессов предоставление услуг и отправки запросов на них на разных компьютерах в сети, каждый из которых выполняют свои задачи независимо от других.

Архитектура программного обеспечения представлена на рисунке 2.1.

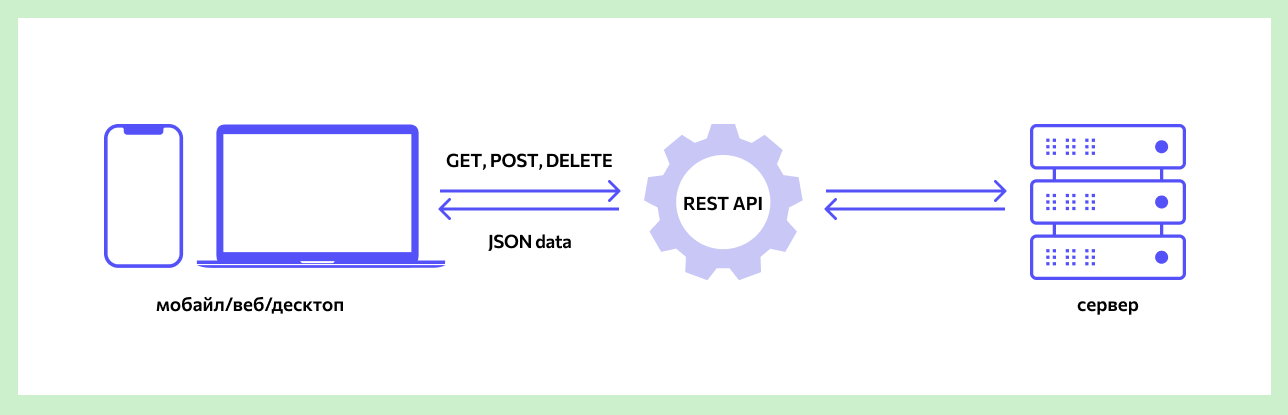


Рисунок 2.1 – Архитектура программного обеспечения

В качестве архитектуры *API* для обработки клиентских запросов используется *RESTful*. *RESTful* относится к программной архитектуре, которая расшифровывается как *Representational State Transfer* (передача репрезентативного состояния). Эта архитектура известна в контексте стандартизации использования систем обмена информацией (веб-сервисов). Этот веб-сервис использует протокол без статического состояния, чтобы сделать текстовые представления онлайн-ресурсов доступными для чтения и обработки. Клиент выполняет действия на основе *HTTP*, такие как получение данных.

Запрос в данной архитектуре содержит четыре компонента:

– конечная точка, являющаяся *URL*-адресом;

– тип метода (*GET*, *POST* и т.д.);

– заголовки, выполняющие функции аутентификации, предоставление информации о содержимом тела;

– данные (или тело).

## **2.2 Топологии нейронных сетей**

Искусственная нейронная сеть – это набор нейронов, соединенных между собой. Как правило, передаточные, активационные функции всех нейронов в сети фиксированы, а веса являются параметрами сети и могут изменяться. Некоторые входы нейронов помечены как внешние входы сети, а некоторые выходы – как внешние выходы сети [4].

Подавая любые числа на входы сети, мы получаем какой-то набор чисел на выходах сети. Таким образом, работа нейросети состоит в преобразовании входного вектора *X* в выходной вектор *Y*, причем это преобразование задается весам сети.

Классифицируя нейронные сети по топологии, можно выделить три основных типа:

1. Полносвязные сети (рис. 2.2, а);

2. Многослойные или слоистые сети (рис 2.2, б)

3. Слабосвязные сети (рис. 2.2, в).

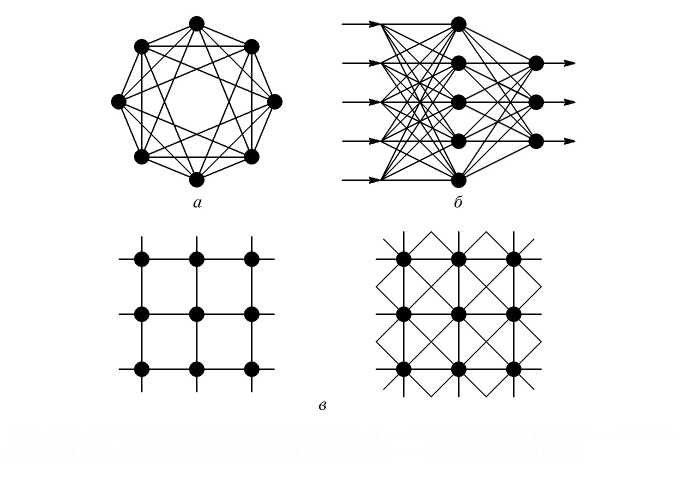
**

Рисунок 2.2 – Топологии нейронных сетей

Полносвязные сети представляют собой искусственные нейронные сети, каждый нейрон которой передает свой выходной сигнал остальным нейронам, в том числе и самому себе (рис. 2-а). Все входные сигналы подаются всем нейронам. Выходными сигналами сети могут быть все или некоторые выходные сигналы нейронов.

В многосвязных (или многослойных) сетях нейроны объединяются в слои. Слой содержит совокупность нейронов с едиными входными сигналами. Число нейронов в каждом слое может быть любым и никак заранее не связано с количеством нейронов в других слоях. Внешние входные сигналы подаются на входы нейронов первого слоя (входной слой часто нумеруют как нулевой), а выходами сети являются выходные сигналы последнего слоя. Вход нейронной сети можно рассматривать как выход «нулевого слоя» вырожденных нейронов, которые служат лишь в качестве распределительных точек, суммирования и преобразования сигналов здесь не производится. Кроме входного и выходного слоев в многослойной нейронной сети есть один или несколько промежуточных (скрытых) слоев.

Связи от выходов нейронов некоторого слоя *q* к входам нейронов следующего слоя (*q* + 1) называются последовательными.

В свою очередь, среди слоистых сетей выделяются несколько типов.

Сети без обратных связей (прямого распределения). В таких сетях нейроны входного слоя получают входные сигналы, преобразуют их и передают нейронам 1-го скрытого слоя, далее срабатывает 1-й скрытый слой и т.д. до *Q*-гo, который выдает выходные сигналы для интерпретатора и пользователя (рис. 2-б). Если не оговорено противное, то каждый выходной сигнал *i*-го слоя подастся на вход всех нейронов (*q*+l)-го слоя; однако возможен вариант соединения *q*-го слоя с произвольным (*q*+*р*)-м слоем.

Сети с обратными связями. Это сети, у которых информация с последующих слоев передается на предыдущие. В качестве примера сетей с обратными связями можно рассматривать так называемые частично-рекуррентные сети Элмана и Жордана.

Слабосвязные сети (нейронные сети с локальными связями) представляют собой слоистые сети с небольшим количеством связей.

**2.3 Архитектура сверточных нейронных сетей**

Свёрточная нейронная сеть (СНС) – класс глубоких нейронных сетей, часто применяемый в анализе визуальных образов. Сверточные нейронные сети являются разновидностью многослойного персептрона с использованием операций свёртки[5, c.128].

В обычном перcептроне, который представляет собой полносвязную нейронную сеть, каждый нейрон связан со всеми нейронами предыдущего слоя, причем каждая связь имеет свой персональный весовой коэффициент. В свёр-точной нейронной сети в операции свёртки используется лишь ограниченная матрица весов небольшого размера, которую «двигают» по всему обрабатываемому слою (в самом начале – непосредственно по входному изображению), формируя после каждого сдвига сигнал активации для нейрона следующего слоя с аналогичной позицией. То есть для различных нейронов выходного слоя используются одна и та же матрица весов, которую также называют ядром свёртки. Её интерпретируют как графическое кодирование какого-либо признака, например, наличие наклонной линии под определенным углом. Тогда следующий слой, получившийся в результате операции свёртки такой матрицей весов, показывает наличие данного признака в обрабатываемом слое и её координаты, формируя так называемую карту признаков [6, c-272].

СНС состоит из разных видов слоев: сверточные (*convolutional*) слои, субдискретизирующие (*subsampling*) слои и слои «обычной» нейронной сети – персептрона.

Архитектура сверточной нейронной сети представлена на рисунке 2.3.

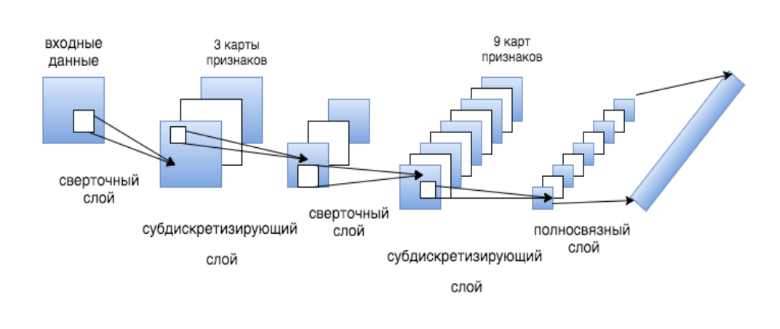


Рисунок 2.3 – Архитектура сверточной нейронной сети

Первые два типа слоев (*convolutional*, *subsampling*), чередуясь между собой, формируют входной вектор признаков для многослойного персептрона.

Операцию свёртки можно представить следующим алгоритмом:

* скользящее окно, называемое фильтром, с размером (*n*, *n*) двигается по входному признаку. Количество движений определяется заданным количеством фильтров.
* Каждый полученный шаблон имеет форму (*n,* *n*, *d*), где *d* – глубина входного признака;
* каждый шаблон умножается на своё ядро свёртки, в результате, формируется выходная карта признаков. Полученная выходная карта признаков имеет форму (*h*, *w*, *N*), где *h* и *w* – длина и ширина, полученные в результате отсечения, а N – количество фильтров.

Количество фильтров – гиперпараметр, поэтому выбирается самостоятельно. Обычно его подбирают как степень двойки с увеличением количества фильтров по мере увеличения глубины архитектуры. А ядра свёртки являются обучаемыми параметрами.

## **2.4 Сравнение фреймворков глубокого обучения**

***2.4.1*** *Tensorflow*,созданная *Google* и написанная на *Python* и *C++,* является одной из лучших открытых библиотек для численных вычислений. Она просто обязана быть хорошей, поскольку даже такие компании как *DeepMind*, *Uber*, *AirBnB* или *Dropbox* выбрали этот фреймворк для своих нужд [4, c.254].

*TensorFlow* хороша для сложных проектов, таких как создание многослойных нейронных сетей. Она используется для распознавания голоса или картинок и приложений для работы с текстом, таких как *Google* *Translate*, например.

Для нее написано большое количество руководств и документации.

Она предлагает мощные средства мониторинга процесса обучения моделей и визуализации (*Tensorboard*), поддерживается большим сообществом разработчиков и техническими компаниями, обеспечивает обслуживание моделей и поддерживает распределенное обучение.

*TensorFlow* *Lite* обеспечивает вывод на устройства с низкой задержкой для мобильных устройств.

*TensorFlow* имеет более высокий входной порог для начинающих, чем *PyTorch* или *Keras*. Голая *Tensorflow* достаточно низкоуровневая и требует много шаблонного кода, и режим «определить и запустить» для *Tensorflow* значительно усложняет процесс дебага.

Есть и еще одно значительное ограничение: единственный полностью поддерживаемый язык – *Python*.

***2.4.2*** *PyTorch*является преемником *Python* для библиотеки *Torch*, написанной на *Lua*, и большим конкурентом *TensorFlow*. Он был разработан *Facebook* и использовался *Twitter,* *Salesforce*, Оксфордским Университетом и многими другими компаниями.

*PyTorch* используется в основном, чтобы обучать модели быстро и эффективно, поэтому это выбор большого количества разработчиков.

У него есть множество важных преимуществ:

* благодаря архитектуре фреймворка, процесс создания модели достаточно прост и прозрачен;
* режим по умолчанию “*define*-*by*-*run*” – отсылка к традиционному программированию. Фреймворк поддерживает популярные инструменты для дебага, такие как *pdb*, *ipdb* или дебаггер *PyCharm*;
* он поддерживает декларативный параллелизм данных;
* он имеет много предварительно обученных моделей и готовых модульных частей, которые легко комбинировать;
* распределенное обучение доступно с версии 0.4.

Но есть у этого фреймворка и несколько неоспоримых минусов:

* недостаточная поддержка моделей;
* недостает интерфейсов для мониторинга и визуализации, как *TensorBoard* – однако он имеет внешнее подключение к *Tensorboard*.

***2.4.3*** *Keras* – это минималистичная библиотека, основанная на *Python*, которая может запускаться поверх *TensoFlow*, *Theano* или *CNTK*. Она была разработана инженером компании *Google*, Франсуа Шолле, в целях ускорения экспериментов. *Keras* поддерживает широкий спектр слоев нейронных сетей, таких как сверточные слои, рекуррентные или плотные [5, c.128].

Этот фреймворк хорош в кейсах для перевода, распознавании изображений, речи и т.п.

Преимущества:

* прототипирование действительно быстрое и простое;
* он достаточно маловесный для построения моделей глубокого обучения для множества слоев;
* имеет полностью конфигурируемые модули;
* имеет простой и интуитивно-понятный интерфейс, соответственно, хорош для новичков;
* имеет встроенную поддержку для обучения на нескольких *GPU*;
* может быть настроен в качестве оценщиков для TensorFlow и обучен на кластерах *GPU* на платформе *Google* *Cloud*;
* запускается на *Spark*;
* поддерживает *GPU* от *NVIDIA*, *TPU* от *Google*, *GPU* с *Open*-*CL*, такие как *AMD*.

И немного недостатков:

* может оказаться слишком высокоуровневым и не всегда легко кастомизируется;
* он ограничен бэкэндами *Tensorflow*, *CNTK* и *Theano*.

*Keras* не настолько функционален как *TensorFlow* и дает меньше опций для управления сетевым соединением, что может стать серьезным ограничением, если вы собираетесь создать какую-то специализированную модель глубокого обучения.

## **2.5 Архитектура *YOLOv5***

*YOLO (You Only Look Once)* – архитектура нейронных сетей, предназначенная для детекции объектов на изображении. Отличительной особенностью *YOLO* является подход к решению задачи детекции [6].

Один из способов решения задачи детекции заключается в разбиении изображения на квадратные области, затем классификация этих областей на наличие объекта и классификация самого объекта. Таким образом, изображение просматривается дважды (один раз для определения областей, где есть объект, второй – для классификации этого объекта.) Этот способ работает долго и требует больших затрат вычислительных мощностей.

*YOLO* же использует другой принцип. Исходное изображение сжимается таким образом, чтобы получить квадратную матрицу размером 13 на 13, в каждой клетке которой записана информация о наличии объекта и классе этого объекта на соответствующей части картинки. Таким образом, *YOLO* просматривает картинку один раз, что существенно увеличивает скорость обработки.

*YOLOv5* – усовершенствованная пятая версия *YOLO*, реализованная на фреймворке *PyTorch* [7].

Архитектура *YOLOv5* предствалена на рисунке 2.4.

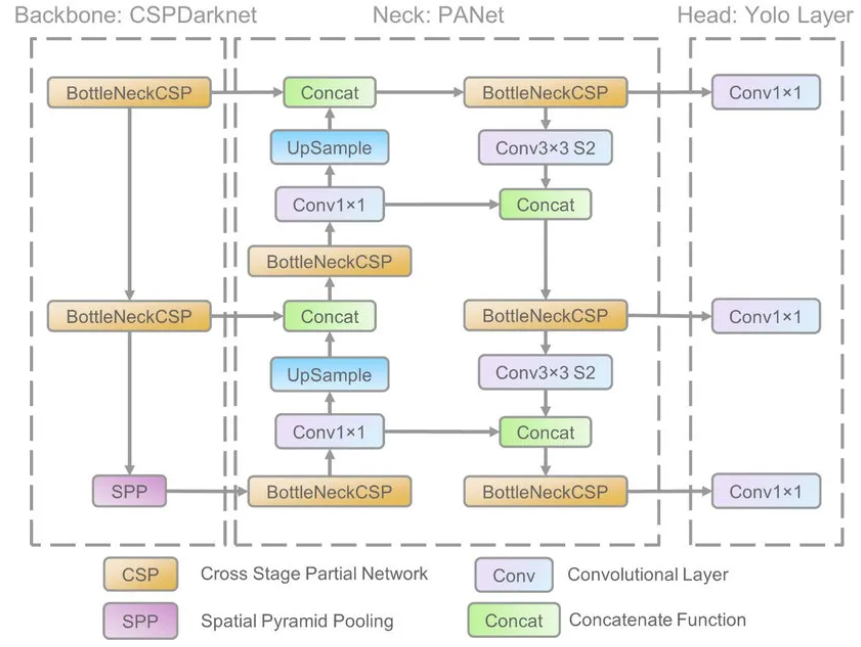


Рисунок 2.4 – Архитектура *YOLOv5*

Семейство моделей YOLO состоит из трех основных архитектурных блоков:

1. *Backbone* – использует *CSPDarknet* в качестве основы для извлечения объектов из изображений, состоящих из многоступенчатых частичных сетей;

2. *Neck* – использует *PANet* для создания сети пирамид объектов для выполнения агрегации объектов и нее передачи в *Head* для прогнозирования;

3. *Head* – cлои, которые генерируют прогнозы по блокам, полученным с помощью *Neck*.

*YOLOv5* имеет множество разновидностей предварительно обученных моделей. Разница между ними заключается в компромиссе между размером модели и временем распознавания. Облегченная версия модели *YOLOv5s* имеет размер всего 14 МБ, но не очень точна. С другой стороны, есть *YOLOv5x*, размер которого составляет 168 МБ, но данный вариант является наиболее точной версией из всех разновидностей.

Разновидности *YOLOv5* представлены на рисунке 2.5.

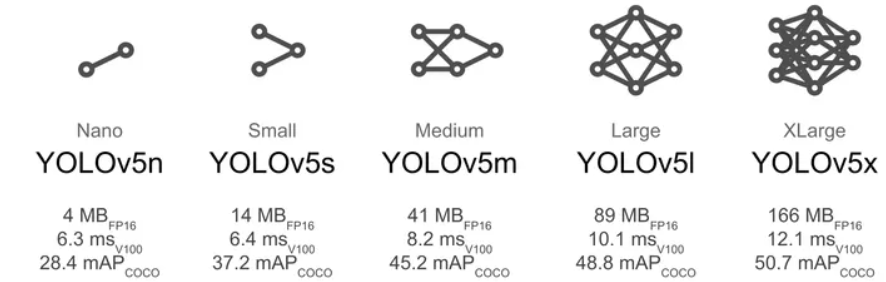


Рисунок 2.5 – Разновидности *YOLOv5*

**3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ**

**3.1 Структура разработанного программного комплекса**

Программный комплекс решающий поставленную задачу представлен двумя независимыми приложениями:

*– RESTful API* реализованное с использованием языка программирования *Python* при использовании фреймворка *Flask*.

– клиент, реализованный с использованием языка программирования *JavaScript* при использовании фреймфорка *React*.

*Flask* – это упрощенная платформа *Python* для веб-приложений, которая обеспечивает основные возможности маршрутизации *URL*-адресов и визуализации страниц. *Flask* называют "микро"-платформой, так как она не предоставляет напрямую такие функции, как проверка форм, абстракция базы данных, проверка подлинности и т. д. Эти функции предоставляются специальными пакетами *Python*, называемыми расширениями *Flask*. Расширения легко интегрируются с *Flask* и отображаются так, как будто являются частью самой среды *Flask*.

*React* – это библиотека *JavaScript* с открытым кодом для создания внешних пользовательских интерфейсов. В отличие от других библиотек *JavaScript*, предоставляющих полноценную платформу приложений, *React* ориентируется исключительно на создание представлений приложений через инкапсулированные единицы (называются компонентами), которые сохраняют состояние и генерируют элементы пользовательского интерфейса.

Также при разработке приложения использовались следующие библиотеки:

– *PyTorch* – фреймворк машинного обучения для языка *Python* с открытым исходным кодом, созданный на базе *Torch*, используется для обучения нейронной сети и выполнения обнаружения объектов;

– Torchvision – это вспомогательная библиотека компьютерного зрения, которая предоставляет функционал для эффективного преобразования изображений и видео, а также некоторые часто используемые предварительно обученные модели;

– *OpenCV* – это библиотека компьютерного зрения, которая предназначена для анализа, классификации и обработки изображений;

– *NumPy* – предоставляет общие математические и числовые операции в виде пре-скомпилированных, быстрых функций.

– *io* – библиотека предоставляющая функционал для работы с потоковыми ресусрами, используется для обработки полученных изображений и отправки результатов в виде изображения.

При проектировании веб-сервиса (*API*) нужно было определить ресурсы, которые будут доступны и запросы, с помощью которых эти данные будут доступны, согласно правил *RESTful*, в качестве ресурса используется модель нейронной сети, а запроса операция на выполнение обнаружения объектов на заданном изображении.

Список доступных конечных точекразработанного *API* представлен в таблице 3.1.

Таблица 3.1 – Описание конечных точек *API*

|  |  |  |
| --- | --- | --- |
| **Метод *HTTP*** | **Путь** | **Описание** |
| *POST* | */v1/object-detection/yolo/json* | Получает изображение и выполняет обнаружение объектов с помощью модели *YOLO* возвращая результаты в формате *JSON* |
| *POST* | */v1/object-detection/yolo/image* | Получает изображение и выполняет обнаружение объектов с помощью модели *YOLO* и возвращает результаты в виде изображения на котором отображены найденные объекты |

Объект возвращаемый при обращении по пути */v1/object-detection/yolo/json* содержит следующие поля:

– *box* – массив из четырех элементов, хранящий информацию о расположении объекта на исходном изображении;

– *class* – название класса к которому принадлежит обнаруженный объект;

– *score* – вероятность принадлежности объекта к у казанному в поле *score* классу.

Пример *JSON* объекта, возвращаемого при обращении к *API* представлен на рисунке 3.1.



Рисунок 3.1 – Пример возвращаемого *JSON* объекта

Пример изображения возвращаемого при обращении к ресурсу */v1/object-detection/yolo/image* представлен на рисунке 3.2.



Рисунок 3.2 – Пример изображения, возвращаемого при обращении по адресу */v1/object-detection/yolo/image*

**3.2 Набор данных**

Предметной областью являются огурцы. Для реализации приложения подготовлен набор изображений огурцов включая другие овощи и фрукты.

Набор данных представлен двумя классами:

– *cucumber* – представляет огурцы;

– *trash* – представляет любой другой овощ либо фрукт чтобы позволить сети узнавать объекты отличные от огурца и, следовательно, уменьшить вероятность того что неизвестный объект будет принят за огурец.

Пример размеченных данных представлен на рисунке 3.3.

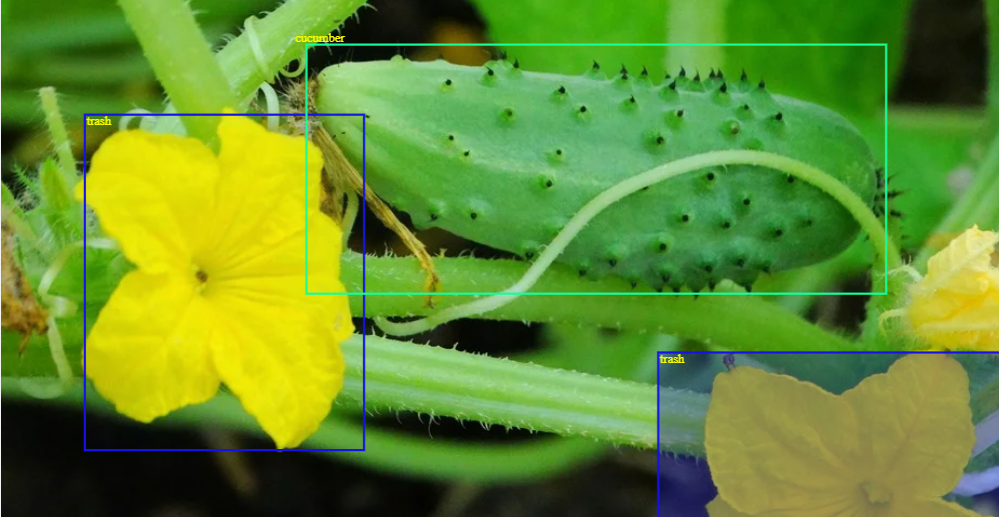
**

Рисунок 3.3 – Пример размеченных данных

Формат набора данных: *YOLO txt* и представляет собой список текстовых файлов, имеющих название изображения и содержащих список отмеченных объектов с указанием номера класса. Формат текстовых файлов представлен на рисунке 3.4.

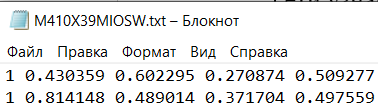


Рисунок 3.4 – Формат текстовых файлов набора данных

* 1. **Обучение модели *YOLOv5***

Для обучения *YOLOv5* выбраны разновидности *nano* и *small* которые показывают наибольшую производительность в нахождении объектов.

Для обучения *YOLOv5* необходимо:

– библиотека *YOLOv5* и вспомогательные *Python* пакеты необходимые для работы с данной библиотекой*;*

– предобученные веса для необходимой разновидности *YOLOv5*;

– файл *yaml* (в нем указываются пути до тренировочной и валидационной выборок, а также количество классов и их метки);

– набор данных.

Основные *Python* пакеты необходимые для работы с библиотекой *YOLOv5*:

– *matplotlib* – используется для создания графиков с информацией о процессе обучения;

– *numpy* – используется для работы с матрицами;

– *opencv*-*python* – используется для работы с изображениями;

– *torch* и *torchvision* – используются для реализации модели.

*yaml* файл необходимый для обучения модели представлен на рисунке 3.5.

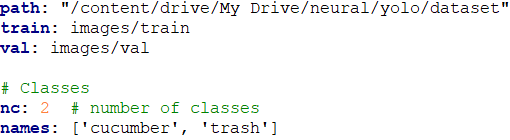


Рисунок 3.5 – *yaml* файл необходимый для обучения модели

В таблице 3.2 находятся результаты обучения модели при разных настройках.

Таблица 3.2 – Результаты обучения модели при разных настройках

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Разновидность | Количество эпох | Размер  подвыборки | *Precision* | *Mean average*  *precision(mAP)* |
| *nano* | 200 | 16 | 0.867 | 0.9 |
| *nano* | 200 | 32 | 0.889 | 0.931 |
| *nano* | 200 | 64 | 0.866 | 0.91 |
| *small* | 200 | 16 | 0.914 | 0.933 |
| *small* | 200 | 32 | 0.95 | 0.956 |
| *small* | 200 | 64 | 0.914 | 0.936 |

В результате обучения наиболее хорошие результаты получены с ипользованием разновидности *small* и размера подвыборки 32. Веса из данного обучения и в следствие и использованы в приложении.

**3.4 Функционал разработанного приложения**

Разработанное приложение запускается из любого веб-браузера. При переходе по адресу приложения открывается страница, показанная на рисунке 3.6.

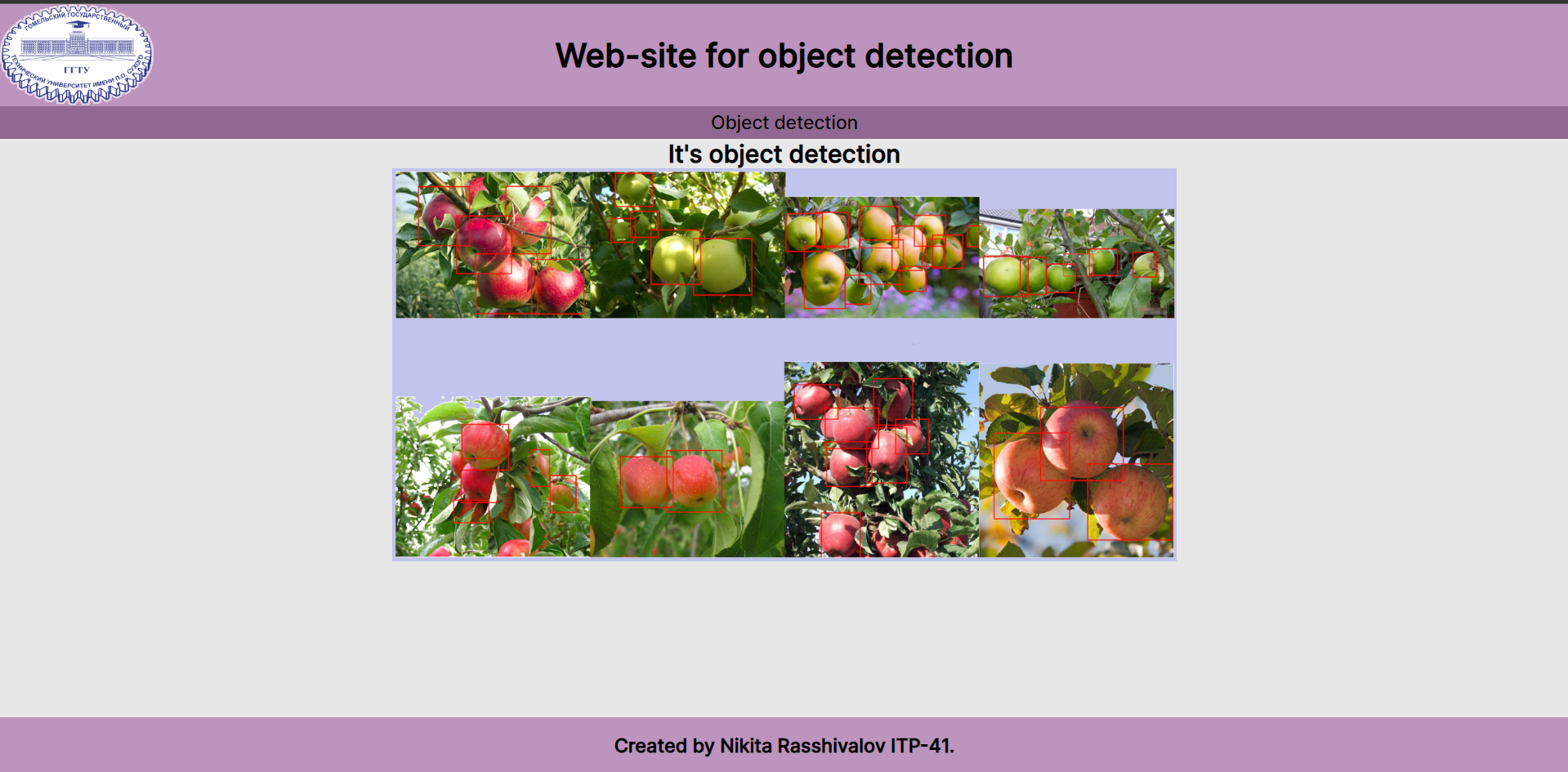


Рисунок 3.6 – Домашняя страница приложения

При нажатии на вкладку “*Object detection*” происходит переход на страницу, которая предоставляет возможность загрузки изображения с компьютера и получения изображения с результатами работы нейронной сети.

Страница вкладки “*Object detection*” представлена на рисунке 3.7.

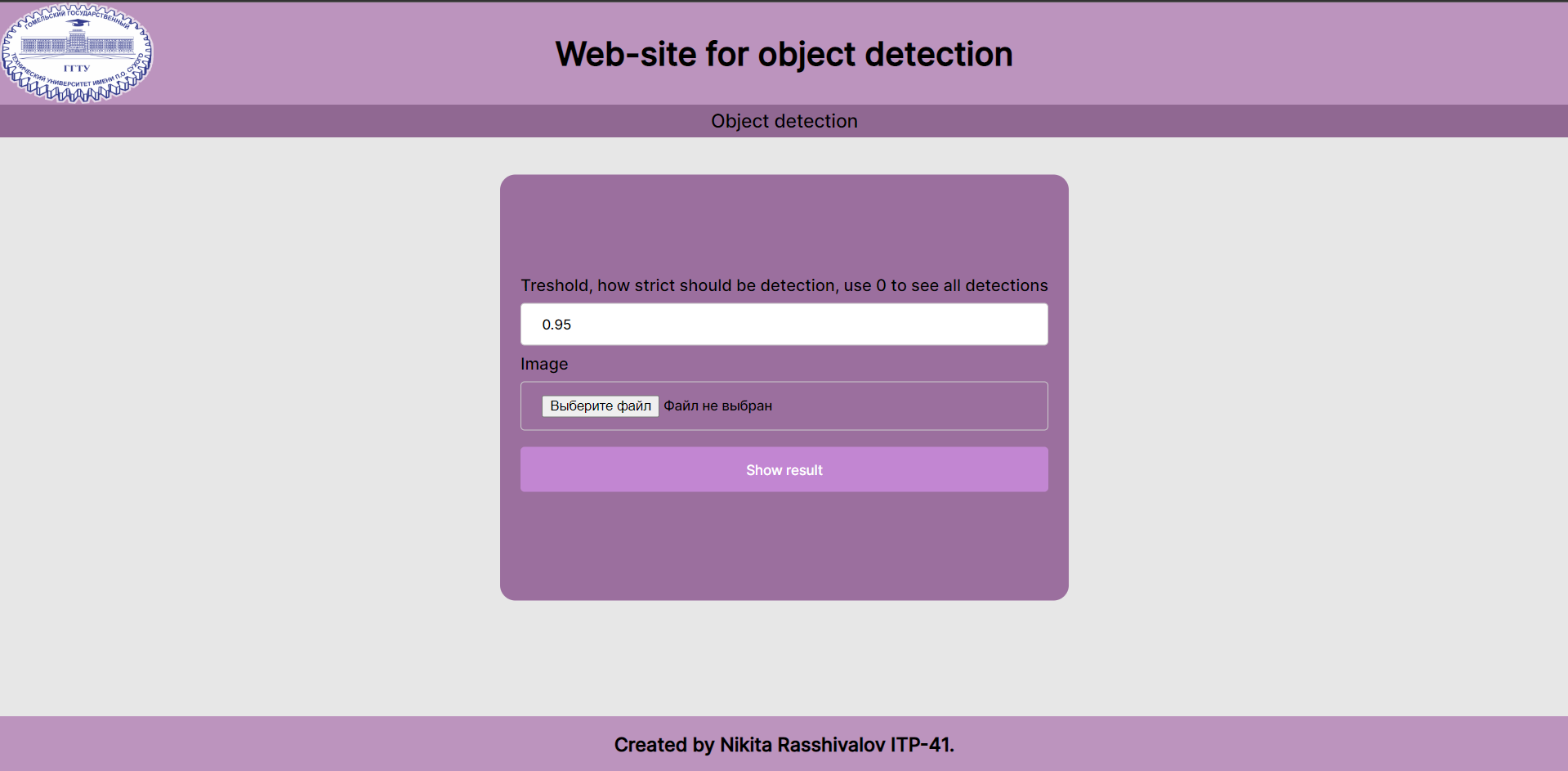


Рисунок 3.7 – Страница вкладки “*Object detection*”

Отображение результата работы нейронной сети после нажатия на кнопку “*Show result*” представлено на рисунке 3.8.



Рисунок 3.8 – Отображение результата работы нейронной сети после нажатия на кнопку “*Show result*”

Также предусмотрена возможность скачивания обработанного файла при нажатии на кнопку “*Download img*”, имя обработанного изображения содержит имя исходного файла с добавлением префикса “*result*\_”.

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсового проекта были изучены и проанализированы данные в области разработки программного обеспечения для решения задачи обнаружения объектов на изображениях при помощи нейронных сетей.

В ходе разработки программного обеспечения решены следующие задачи:

– изучены средства и подходы разработки клиент-серверных приложений;

– изучены и применены паттерны проектирования для создания *API,* которое может быть использовано у множества различных клиентов;

– написан программный код, производящий обнаружение объектов на изображениях с последующим отображением в веб-браузере;

– проведена верификация с использованием метрик качества работы обученной модели нейронной сети*.*

Программный комплекс разработан на языках программирования *Python* и *JavaScript*, с использованием фреймворков *Flask* и *React*.

Приложение имеет высокую скорость для подсчета данных и точность предоставляемых результатов, позволяя пользователям производить быстрое обнаружение объектов на изображениях.

# **Список использованных источников**

1. Саймон, Х. Нейронные сети. Полный курс: Пер. с англ. / Х. Саймон – СПб: Питер, 2018. – 1104 с.
2. Шолле, Ф. Глубоко обучение на *Python*: Пер. с англ. / Ф. Шолле – СПб: Питер, 2018. – 400 с.
3. *Introduction to YOLOv5 object detection* [Электронный ресурс]. – 2022. – Режим доступа: https://machinelearningknowledge.ai/introduction-to-yolov5-object-detection-with-tutorial/. – Дата доступа: 10.11.2022.
4. Николенко, С. Глубокое обучение / С. Николаенко – СПб: Питер, 2018. – 481 с.
5. Сверточные нейронные сети [Электронный ресурс]. – 2016. – Режим доступа: https://habr.com/ru/post/309508. – Дата доступа: 11.11.2022.
6. Самарасингхе, С. Нейронные сети для прикладных наук и инженерии: от основ до распознавания сложных образов / С. Самарасингхе – СПб: Питер, 2007. – 374 с.
7. *YOLOv5* [Электронный ресурс]. – 2022. – Режим доступа: https://github.com/ultralytics/yolov5. – Дата доступа: 15.11.2022.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Листинг программы**

!git clone https://github.com/ultralytics/yolov5 # clone

#!git reset --hard cd35a009ba964331abccd30f6fa0614224105d39

%cd yolov5

%pip install -qr requirements.txt # install

import torch

import utils

display = utils.notebook\_init() # checks

!python train.py --img 640 --batch 24 --epochs 100 --data "/content/drive/My Drive/neural/yolo/datasetv4.1/cucumbers.yaml" --weights yolov5s.pt –cache

!python export.py --weights runs/train/exp4/weights/best.pt --include torchscript

!zip -r /content/sample\_databest.zip /content/yolov5/runs/train/exp/weights

from google.colab import files

files.download('/content/sample\_databest.zip')

import flask

import pandas as pd

import torch

import argparse

import io

from flask import request,send\_file, make\_response,jsonify

import cv2

from PIL import Image

import numpy as np

WEIGHT\_DIR = "E:/University/7sem/OPR/Course/data/best.pt"

#model

def create\_model():

model = torch.hub.load('yolov5', 'custom',path=r'E:/University/7sem/OPR/Course/data/best.pt', source='local')

return model

def do\_detection(image):

with torch.no\_grad():

outputs = model(image).pandas().xyxy[0]

return outputs

def get\_image\_from\_bytes(binary\_image, max\_size=1024):

input\_image =Image.open(io.BytesIO(binary\_image)).convert("RGB")

width, height = input\_image.size

resize\_factor = min(max\_size / width, max\_size / height)

resized\_image = input\_image.resize((

int(input\_image.width \* resize\_factor),

int(input\_image.height \* resize\_factor)

))

return resized\_image

#initialize model data

CLASSES = ['background', 'cucumber']

DEVICE = torch.device('cuda') if torch.cuda.is\_available() else torch.device('cpu')

detection\_threshold = 0.95

app = flask.Flask(\_\_name\_\_)

@app.route("/health")

def health():

return "ok"

@app.route("/v1/object-detection/yolo/image", methods=["POST", "OPTIONS"]) # type: ignore

def yolo\_to\_image():

if request.method == "OPTIONS": # CORS preflight

return \_build\_cors\_preflight\_response()

if not request.method == "POST":

return

threshold = detection\_threshold

threshold\_parameter = request.args.get('threshold')

if threshold\_parameter is not None:

threshold = float(threshold\_parameter)

if request.files.get("file"):

model.conf = threshold

image\_file = request.files["file"]

image\_bytes = image\_file.read()

image = Image.open(io.BytesIO(image\_bytes))

results = model([image],size=640)

results.render() # updates results.imgs with boxes and labels

for img in results.ims:

bytes\_io = io.BytesIO()

img\_base64 = Image.fromarray(img)

img\_base64.save(bytes\_io, format="png")

return \_corsify\_actual\_response(send\_file(io.BytesIO(bytes\_io.getvalue()), mimetype="image/png"))

@app.route("/v1/object-detection/yolo/json", methods=["POST", "OPTIONS"]) # type: ignore

def yolo\_detection\_to\_json():

if request.method == "OPTIONS": # CORS preflight

return \_build\_cors\_preflight\_response()

if not request.method == "POST":

return

threshold = detection\_threshold

threshold\_parameter = request.args.get('threshold')

if threshold\_parameter is not None:

threshold = float(threshold\_parameter)

if request.files.get("file"):

image\_file = request.files["file"]

image\_bytes = image\_file.read()

image = Image.open(io.BytesIO(image\_bytes))

results = model([image],size=640)

results.render() # updates results.imgs with boxes and labels

return \_corsify\_actual\_response(jsonify(results.pandas().xyxy[0].to\_json(orient="records")))

def \_build\_cors\_preflight\_response():

response = make\_response()

response.headers.add("Access-Control-Allow-Origin", "\*")

response.headers.add('Access-Control-Allow-Headers', "\*")

response.headers.add('Access-Control-Allow-Methods', "\*")

return response

def \_corsify\_actual\_response(response):

response.headers.add("Access-Control-Allow-Origin", "\*")

return response

parser = argparse.ArgumentParser(description="Flask api")

parser.add\_argument("--port", default=5010, type=int, help="port number")

args = parser.parse\_args()

model = create\_model()

model = model.to(DEVICE) # type: ignore

model.eval()

app.run(host="127.0.0.1", port=args.port,debug=True)

**Client**

import React, { useState } from 'react';

import { ObjectDetectionService } from '../../services/ObjectDetectionService';

export function ObjectDetection() {

const [threshold, setTreshold] = useState(0.95);

function handleTresholdChange(e){

if(/^[0-9]\d\*\.?\d\*$/.test(e.target.value)){

setTreshold(e.target.value);

}

}

const [imagesBlockClass, setImagesBlockClass] = useState("images-block");

const [currentFileName, setCurrentFileName] = useState("result.png");

const [imageSrc, setImageSrc] = useState(null);

const [image, setImage] = useState(null);

function handleImageChange(e){

if(e.target.files.length === 0){

return;

}

if(e.target.files.length > 1){

alert("Cannot attach more then one file")

}

let file = e.target.files[0];

if (file) {

setCurrentFileName("result\_" + file.name)

setImage(file);

}

}

function handleSubmit(){

let service = new ObjectDetectionService();

service.processImage(image, threshold).then((newImageSrc) => {

if(newImageSrc !== null)

{

setImageSrc(newImageSrc);

setImagesBlockClass(imagesBlockClass + " " + "images-block-active");

}

});

}

return (

<>

<div className="all-container">

<div className="form">

<label>Treshold, how strict should be detection, use 0 to see all detections</label>

<input onChange={handleTresholdChange} value={threshold} pattern="^[0-9]\d\*\.?\d\*$" type="text" className="form-input" maxLength="8" placeholder="Treshold.." />

<label>Image</label>

<input onChange={handleImageChange} accept="image/png, image/jpeg" type="file" className="form-input" />

<button className="form-submit" onClick={handleSubmit}>Show result</button>

</div>

</div>

<div className={imagesBlockClass}>

<div className="grid-item">

<img src={imageSrc} width="245" height="356" />

</div>

<div className="grid-item">

<a download={currentFileName} className="details-button" href={imageSrc}>Download image</a>

</div>

</div>

</>

);

}

import React from 'react';

import detectPhoto from '../../content/detect.jpg'

export function Home() {

return (

<div className='home-content'>

<h2>It's object detection</h2>

<div className='content-img'>

<img src={detectPhoto}></img>

</div>

</div>

);

}

import React, { Component } from 'react';

export class Footer extends Component {

static displayName = Footer.name;

render () {

return (

<footer className="footer">

<div className="footer-item">

<h3>Created by Nikita Rasshivalov ITP-41.</h3>

</div>

</footer>

);

}

}

import React, { Component } from 'react';

import {AppRoutes} from '../../constants/AppRoutes';

import logo from '../../content/logo.png'

export class Header extends Component {

static displayName = Header.name;

render () {

return (

<header className="header">

<div className="header-item">

<a href='/'><img className='logo-img' src={logo}></img></a>

<div className="header-title">

<h1>Web-site for object detection</h1>

</div>

</div>

<nav className="navbar">

<a href={AppRoutes.objectDetection}>Object detection</a>

</nav>

</header>

);

}

}

import React, { Component } from 'react';

import { Header } from './Header';

import { Footer } from './Footer';

export class Layout extends Component {

static displayName = Layout.name;

render () {

return (

<>

<div className="content">

<Header />

{this.props.children}

</div>

<Footer />

</>

);

}

}

export const AppRoutes = {

home: '/',

objectDetection: '/objectDetection',

}

export class ObjectDetectionService {

objectDetectionApiHost = "http://127.0.0.1:5010/";

async processImage(image, threshold = 0.95) {

try {

var data = new FormData()

data.append('file', image)

let response = await fetch(this.objectDetectionApiHost + `/v1/object-detection/yolo/image?threshold=${threshold}`, {

method: "POST",

body: data

})

let imageBlob = await response.blob();

return await this.blobToBase64(imageBlob);

} catch {

}

return null;

}

blobToBase64(blob) {

return new Promise((resolve, \_) => {

const reader = new FileReader();

reader.onloadend = () => resolve(reader.result);

reader.readAsDataURL(blob);

});

}

}

import './App.css';

import React from 'react';

import {

BrowserRouter,

Routes,

Route,

} from "react-router-dom";

import {Home} from './components/home/Home';

import {ObjectDetection} from './components/detection/ObjectDetection';

import {AppRoutes} from './constants/AppRoutes';

export function App() {

return (

<BrowserRouter>

<Routes>

<Route path={AppRoutes.home} element={<Home />} />

<Route path={AppRoutes.objectDetection} element={<ObjectDetection />} />

</Routes>

</BrowserRouter>

);

}

export default App;

import React from 'react';

import ReactDOM from 'react-dom/client';

import App from './App';

import { Layout } from './components/shared/Layout';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

<React.StrictMode>

<Layout>

<App />

</Layout>

</React.StrictMode>

);

// If you want to start measuring performance in your app, pass a function

// to log results (for example: reportWebVitals(console.log))

// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals

**ПРИЛОЖЕНИЕ Б**

(Обязательное)

**Схема алгоритма**