**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

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

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

по дисциплине «Программирование сетевых приложений»

на тему: **«клиент-серверное приложение, реализующее многопользовательский чат на основе протокола http»**

Исполнитель: студент гр. ИТП-41

Солодков М.А

Руководитель: доцент

Курочка К.С.

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

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

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

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

Подписи   членов   комиссии

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

Гомель 2020

Установа адукацыі «Гомельскі дзяржаўны   
тэхнічны ўніверсітэт імя П.В. Сухога»

Факультэт аўтаматызаваных і інфармацыйных сістэм

Кафедра \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**РЭЦЭНЗІЯ**

**на курсавы праект (работу)**

па дысцыпліне \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ,

выканана студэнтам \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

групы \_\_\_\_\_\_\_\_\_\_

I. Пералік заўваг па тэксту курсавога праекта (работы)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

II. Агульная характарыстыка работы

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

III. Складаючыя агульнай адзнакі па праекту (рабоце)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

1. Своечасовасць выканання

|  |  |  |  |
| --- | --- | --- | --- |
| + |  |  | – |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| нізкі ўзровень | | | |  |  | высокі ўзровень | | | |
|  |  |  |  |  |  |  |  |  |  |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

2. Адпаведнасць заданню

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| нізкі ўзровень | | | |  |  | высокі ўзровень | | | |
|  |  |  |  |  |  |  |  |  |  |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

3. Рацыянальнасць праектных рашэнняў

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| нізкі ўзровень | | | |  |  | высокі ўзровень | | | |
|  |  |  |  |  |  |  |  |  |  |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

4. Правільнасць выканання разлікаў

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| нізкі ўзровень | | | |  |  | высокі ўзровень | | | |
|  |  |  |  |  |  |  |  |  |  |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

5. Поўнасць выканання і якасць

афармлення тлумачальнай запіскі

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| нізкі ўзровень | | | |  |  | высокі ўзровень | | | |
|  |  |  |  |  |  |  |  |  |  |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

6. Поўнасць выканання і якасць

афармлення графічнай часткі (пры наяўнасці)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| нізкі ўзровень | | | |  |  | высокі ўзровень | | | |
|  |  |  |  |  |  |  |  |  |  |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

7. Поўнасць і абгрунтаванасць

высноў па праекту (рабоце)

Адзнака пра допуск

праекта (работы) да абароны \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата допуску, подпіс аднаго з чальцоў камісіі па праверцы)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| нізкі ўзровень | | | |  |  | высокі ўзровень | | | |
|  |  |  |  |  |  |  |  |  |  |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

8. Абарона курсавога праекта (работы)

**Агульная адзнака праекта (работы)** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(выстаўляецца з улікам усіх складнікаў па п.п. 1…8)

« \_\_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_.

(дата)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпісі, ініцыялы, П., І., І.п.б. чальцоў камісіі па праверцы)

**СОДЕРЖАНИЕ**

[Введение 5](#_Toc58808391)

[1 Анализ предметной области 6](#_Toc58808392)

[1.1 Цели детектирования объектов на изображениях 6](#_Toc58808393)

[1.2 Классификация и выделение объектов на изображениях 8](#_Toc58808394)

[1.3 Архитектура клиент-сервер 10](#_Toc58808395)

[2 Разработка математической и информационной модели задачи 13](#_Toc58808396)

[2.1 Задача обнаружение объектов 13](#_Toc58808397)

[2.2 Архитектура сети *R-CNN* 14](#_Toc58808398)

[2.3 Архитектура приложения 18](#_Toc58808399)

[2.4 Серверная часть приложения 19](#_Toc58808400)

[2.5 Клиентская часть приложения 20](#_Toc58808401)

[3 Разработка программного обеспечения для классификации объектов 22](#_Toc58808402)

[3.1 Средства разработки программного обеспечения 22](#_Toc58808403)

[3.2 Пользовательский интерфейс приложения 24](#_Toc58808404)

[3.3 Верификация программного обеспечения 27](#_Toc58808405)

[Заключение 31](#_Toc58808406)

[Список используемых источников 32](#_Toc58808407)

[Приложение А Функциональная схема приложения 33](#_Toc58808408)

[Приложение Б Код программы 34](#_Toc58808409)

# **ВВЕДЕНИЕ**

Вопросы общения интересовали людей всегда. Для того что бы иметь возможность обмениваться какой-либо информацией не только при личной встрече, но и на очень больших расстояниях люди изобретали все новые и новые средства, такие как различные почтовые системы, протягивали кабели через континенты и океаны, запускали спутники связи. Однако это все было не совсем рационально, так как было достаточно дорого для создания и обслуживания.

С развитием информационных технологий стали возможными еще более глобальные коммуникации. Историческим «докомпьютерным» предшественником чата, несомненно, был телефон. Ни почта, ни телеграф не позволяли общаться в режиме реального времени и не были доступны в домашней обстановке.

Изобретение телефона и его распространение как средства связи вызвало настоящуюю революцию в средствах и способах общения. Возможность поговорить с собеседником на другой стороне Земли тогда казалось настоящим чудом.

# **1** **АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

## **1.1 Цели** **детектирования объектов на изображениях**

Цель детектирования – определить наличие объекта на изображении и найти его положение в системе координат пикселей исходного изображения. Положение объекта в зависимости от выбора алгоритма детектирования может определяться координатами прямоугольника, окаймляющего объект, либо контуром этого объекта, либо координатами точек, наиболее характерных для объекта.

Решение задачи детектирования объектов позволяет анализировать качественный состав сцены, представленной на изображении, а также получить информацию о взаимном расположении объектов.

Множество всех методов решения задачи детектирования можно разделить на три основные группы:

– методы, которые для описания объекта используют признаки, наиболее характерные для объектов. В качестве признаков могут быть выбраны точечные особенности объекта, либо признаки, построенные для изображения, содержащего только объект;

– методы поиска объектов, соответствующих шаблону – некоторому описанию объектов;

– методы детектирования движения объектов – выделение движущихся объектов на основании нескольких изображений или кадров видео одной и той же сцены.

Один из возможных подходов к решению задачи детектирования состоит в том, чтобы использовать алгоритмы машинного обучения для построения моделей классов объектов и алгоритмы вывода для поиска объектов на изображении.

Построение модели состоит из двух этапов:

– извлечение признаков, характерных для объектов класса, – построение характеристических векторов-признаков для ключевых точек объекта (углов, ребер или контуров объектов) или для всего объекта;

– тренировка модели на полученных признаках для последующего распознавания объектов.

Техники данной группы описывают объект с использованием векторов-признаков. Вектора строятся на основании цветовой информации. Также может быть использована контекстная информация (*context based*), а в некоторых случаях – данные о геометрии и взаимном расположении частей объекта (*part-based*). Тем не менее, все эти методы строят некоторую математическую модель объекта на каждом изображении тренировочной выборки, содержащем объект. Формально признак – это числовая характеристика. Для каждой ключевой точки алгоритмы данной группы строят вектор признаков (). Таким образом, объект описывается набором векторов признаков в характерных точках. В результате тренировки строится модель, содержащая "усредненные" вектора признаков.

Пример построения модели изображён на рисунке 1.1.

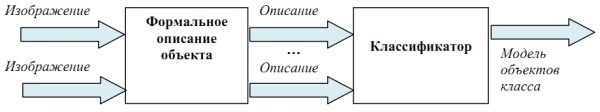


Рисунок 1.1 – Схема построения модели класса с использованием методов, основанных на извлечении характерных признаков

Алгоритм вывода (поиска) включает два этапа:

– извлечение признаков объекта из тестового изображения;

– поиск объектов на изображении, рисунок 1.2.

Входными данными алгоритма поиска являются формальное описание объекта – набор признаков, которые выделены из тестового изображения, – и модель класса объектов. На основании этой информации классификатор принимает решение о принадлежности объекта классу. Некоторые методы поиска также оценивают степень достоверности того, что объект принадлежит рассматриваемому классу.

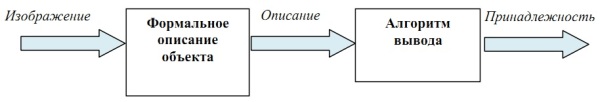


Рисунок 1.2 – Схема поиска объектов с использованием методов, основанных на извлечении характерных признаков

Качество рассматриваемых методов в основном зависит от того, насколько хорошо выбраны признаки, т.е. насколько хорошо эти признаки дифференцируют классы объектов. Существуют специализированные методы, основанные на извлечении признаков, для детектирования лиц, транспортных средств и пешеходов.

## **1.2 Классификация и выделение объектов на изображениях**

Множество практических задач от автоматизации контроля на производстве до конструирования роботизированных автомобилей непосредственно связаны с задачей поиска объектов на изображении. Для её решения можно применять две разные стратегии, которые зависят от условий съёмки – моделирование фона и моделирование объекта [2].

Моделирование фона – этот подход можно применять если камера неподвижна, т.е. мы имеем фон, который мало изменяется, и таким образом можно построить его модель. Все точки изображения, которые существенно отклоняются от модели фона, считаем объектами переднего плана. Таким образом можно решать задачи обнаружения и сопровождения объекта.

Моделирование объекта – этот подход более общий, применяется в случаях, когда фон постоянно и существенным образом изменяется. В отличии от предыдущего случая, здесь нам необходимо знать, что именно мы хотим найти, т.е. необходимо построить модель объекта, а затем проверить точки картинки на соответствие этой модели.

Иногда условия задачи позволяют комбинировать оба подхода, это может существенно улучшить результаты.

Если рассматривать задачу поиска объектов на видео, то её можно разделить на три подзадачи - обнаружение, распознавание и сопровождение.

Машинное обучение можно применять, по крайней мере, для решения задач обнаружения и распознавания, которые сводятся к построению классификатора изображений.

Классификатор изображений состоит из двух частей: метод извлечения признаков (feature extractor) и собственно классификатор.

Методы извлечения признаков из изображения:

***1.2.1*** Гистограммы направленных градиентов (*Histogram of Oriented Gradients, HOG*) – метод извлечения признаков из изображений, который очень похож на метод вычисления дескрипторов для особых точек, только вычисляем его не для окрестности особой точки, но для всего изображения. Общая схема вычисления *HOG* выглядит следующим образом. Картинка разделяется на части (ячейки), для каждой ячейки строим гистограмму направлений градиента яркости, далее гистограммы ячеек нормируются по контрасту и объединяются.

***1.2.2*** Мешок слов (*Bag of visual Words, BoW*) – метод извлечения признаков из изображений, который является адаптацией для изображений метода частотного анализа текстов. Полученные кластеры дескрипторов будут играть роль визуальных слов и составлять словарь для частотного анализа изображения.

***1.2.3*** Признаки Хаара – вычисляется следующим образом: выбирается прямоугольная область на изображении, разбитие её на несколько смежных прямоугольных частей, в каждой части суммируется яркость точек, после вычисляется разность между этими суммами. На основе признаков Хаара построен метод Виолы-Джонса, который часто используют для локализации изображения лиц на фотографиях. Он объединяет в себе признаки Хаара, классификатор *AdaBoost* и метод скользящего окна для разделения картинки на части, рисунок 1.3.

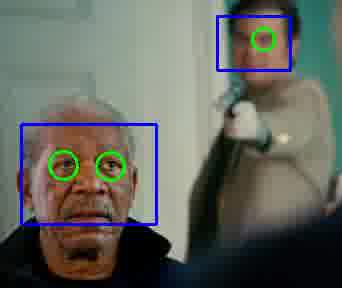
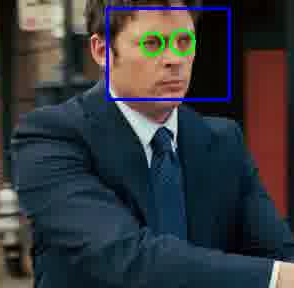


Рисунок 1.3 – Результат работы детектора лиц Виолы-Джонса

***1.2.4*** Свёрточная искусственная нейронная сеть (*convolutional neural network, CNN*) – особенностью этой модели классификатора есть встроенный механизм извлечения признаков из изображения. Он является частью сети и "самонастраивается" в процессе обучения сети.

***1.2.5*** Локализация объектов – результатом работы алгоритма является список *R*, который состоит из позиции окна, соответствующих ему значения масштаба изображения и значения выхода классификатора, рисунок 1.4. Далее необходимо выполнить обработку результатов, которая сводится к приведению координат окон к единому масштабу и удаления лишних (дубликатов) окон [2].

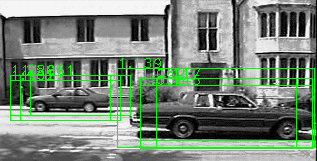


Рисунок 1.4 – Неочищенный результат работы детектора

Удалить дубликаты окон можно выполнив кластеризацию центров этих окон, т.е. близко расположенные (или вложенные) окна образуют свой кластер, рисунок 1.5. В каждом таком кластере выбирается одно окно с максимальным рейтингом (ответом) классификатора.

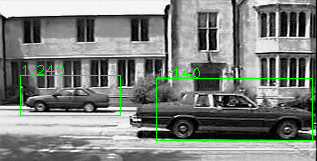


Рисунок 1.5 – Очищенный результат работы детектора

## **1.3 Архитектура клиент-сервер**

К добавлению внешнего сервера рано или поздно приходит любой сложный проект. Причины, при этом, бывают совершенно различные. Одни, загружают дополнительные сведения из сети, другие, синхронизируют данные между клиентскими устройствами, третьи- переносят логику выполнения приложения на сторону сервера. Как правило, к последним относятся большинство «деловых» приложений. По мере отхода от парадигмы «песочницы», в которой все действия выполняются только в рамках исходной системы, логика выполнения процессов переплетается, сплетается, завязывается узлами настолько, что становится трудно понять, что является исходной точкой входа в процесс приложения. В этом момент, на первое место выходит уже не функциональные свойства самого приложения, а его архитектура, и, как следствие, возможности к масштабированию.

Архитектура клиент-сервер предъявляет специфические требования, как к клиенту, так и к серверу. Программа, удовлетворяющая этим требованиям, может считаться клиент-серверным приложением, выполняющим распределенную обработку данных. Под распределенной обработкой понимается выполнение серверной частью программы запросов клиентской части. Серверная часть приложения обеспечивает хранение данных и их обработку, и отправку данных клиенту, а клиентская часть передает серверу соответствующие запросы.

К преимуществам клиент-серверных систем можно отнести:

– клиент-серверный подход – модульный, причем серверные программные компоненты компактны и автономны;

– поскольку каждый компонент выполняется в отдельном защищенном процессе пользовательского режима, сбой сервера не повлияет на остальные компоненты операционной системы;

– автономность компонентов делает возможным их выполнение на нескольких процессорах на одном компьютере (симметричная многопроцессорная обработка) или на нескольких компьютерах сети (распределенные вычисления);

– обязанность клиента, как правило, – предоставлять пользовательские сервисы и, прежде всего, пользовательский интерфейс, то есть средства для приема, отображения и редактирования данных, введенных пользователем, которые служат основой для запроса серверу. Кроме того, клиент можно настроить на обработку части данных, чтобы уменьшить нагрузку на ресурсы сервера.

Недостатками являются:

– неработоспособность сервера может сделать неработоспособной всю вычислительную сеть. Неработоспособным сервером следует считать сервер, производительности которого не хватает на обслуживание всех клиентов, а также сервер, находящийся на ремонте, профилактике и т.п.;

– поддержка работы данной системы требует отдельного специалиста системного администратора;

– высокая стоимость оборудования.

В любой сети, построенной на современных сетевых технологиях, присутствуют элементы клиент-серверного взаимодействия, чаще всего на основе двухзвенной архитектуры. Двухзвенной она называется из-за необходимости распределения трех базовых компонентов между двумя узлами (клиентом и сервером).

Двухзвенная архитектура используется в клиент-серверных системах, где сервер отвечает на клиентские запросы напрямую и в полном объеме, при этом используя только собственные ресурсы. Т.е. сервер не вызывает сторонние сетевые приложения и не обращается к сторонним ресурсам для выполнения какой-либо части запроса.

Еще одна тенденция в клиент-серверных технологиях связана с все большим использованием распределенных вычислений. Они реализуются на основе модели сервера приложений, где сетевое приложение разделено на две и более частей, каждая из которых может выполняться на отдельном компьютере. Выделенные части приложения взаимодействуют друг с другом, обмениваясь сообщениями в заранее согласованном формате. В этом случае двухзвенная клиент-серверная архитектура становится трехзвенной. Как правило, третьим звеном в трехзвенной архитектуре становится сервер приложений. Пример трехзвенной архитектуры представлен на рисунке 1.6.

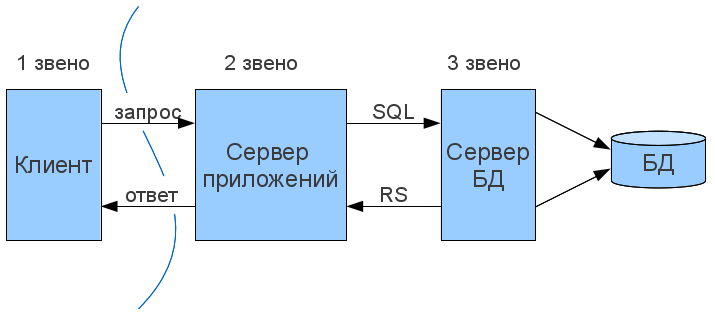


Рисунок 1.6 – трехзвенная клиент-серверная архитектура

Трехзвенная архитектура может быть расширена до многозвенной путем выделения дополнительных серверов, каждый из которых будет представлять собственные сервисы и пользоваться услугами прочих серверов разного уровня.

Со стороны клиента нет никакой разницы между двухзвенной и трезвенной архитектурой. Здесь важно понимание двух вещей:

– может быть множество клиентов, использующих один аккаунт для общения с севером;

– каждый клиент, как правило, имеет свое собственное локальное хранилище.

В данной работе была использована двухзвенная архитектура для решения поставленной задачи. Однако следует отметить, то, что в данном приложении небыли использованы системы управления базами данных, так как в данном случае их использование было бы избыточным. Хоть и допустимым.

В ряде случаев, локальное хранилище может быть синхронизировано с облаком, и, соответственно, с каждым из клиентов.

Следует отметить, что поскольку, некоторые разработчики стремятся избавиться от «серверной части» некоторые приложения построены вокруг синхронизации их хранилищ в «облаке». Т. е. фактически, имеют так же, двухзвенную систему, но с переносом архитектуры её развертывания на уровень операционной системы. В некоторых случаях такая структура оправдана, но такая система не так легко масштабируется, и её возможности весьма ограничены.

На самом примитивном уровне абстракции приложение, ориентированное на работу с сервером, состоит из следующих архитектурных слоев:

– ядро приложения;

– графический пользователь интерфейс;

– компоненты повторного использования;

– файлы окружения;

– ресурсы приложения.

# 2 РАЗРАБОТКА МАТЕМАТИЧЕСКОЙ И ИНФОРМАЦИОННОЙ МОДЕЛИ ЗАДАЧИ

## **2.1** **Задача обнаружение объектов**

Технологии компьютерного зрения позволяют в сегодняшних реалиях сделать жизнь и бизнес проще, дешевле, безопаснее. По оценкам разных экспертов этот рынок будет двигаться в ближайшие годы только в сторону роста, что и позволяет развиваться соответствующим технологиям как в сторону производительности, так и качества. Одним из наиболее востребованных разделов является *Object Detection* (обнаружение объектов) – определение объекта на изображении или в видео потоке.

С решением данной задачи, встречаются часто в компьютерном зрении, где требуется производить аннотирования изображений, распознавать активности, лица, либо производить сегментацию видеообъектив. Так же данную задачу решают для отслеживания объектов, например, для отслеживания мяча во время футбольного матча, отслеживания движения биты для крикета или отслеживания человека на видео.

У каждого класса объектов есть свои отличия, которые помогают в классификации класса – например, все круги круглые. Обнаружение класса объекта использует эти специальные функции. Аналогичный подход используется для идентификации лица, где можно найти глаза, нос и губы, а такие особенности, как цвет кожи и расстояние между глазами.

Методы обнаружения объектов обычно относятся либо к подходам, основанным на машинном обучении, либо к подходам на основе глубокого обучения. Для подходов к машинному обучению становится необходимым сначала определить отличия, используя один из методов для этого, а затем использовать такую методику, как машина опорных векторов (*SVM*) для выполнения классификации. С другой стороны, методы глубокого обучения могут выполнять сквозное обнаружение объектов без специального определения функций и обычно основаны на сверхточных нейронных сетях (*CNN*).

Подходы к машинному обучению:

– среда обнаружения объектов Виолы – Джонса на основе функций Хаара;

– масштабно-инвариантное преобразование признаков (*SIFT*);

– гистограммы ориентированных градиентов (*HOG*).

Подходы к глубокому обучению:

– сверточные нейронные сети на основе регионов (*Region Based Convolutional Neural Networks*), сети, такие как *R-CNN*, *Fast R-CNN*, *Faster R-CNN*, *cascade R-CNN*;

– *Single Shot MultiBox Detector (SSD)*;

– «Ты смотришь только один раз» (*You Only Look Once, YOLO*);

– нейронная сеть однократного уточнения для обнаружения объектов (*RefineDet, Detina-Net*);

– деформируемые свёрточные сети.

## **2.2 Архитектура сети *R-CNN***

Сообщество машинного зрения быстро улучшило результаты обнаружения объектов и семантической сегментации за короткий период времени. В значительной степени эти достижения были вызваны мощными базовыми системами, такими как структуры *Fast* / *Faster R-CNN* и полностью сверточная сеть (*Fully Convolutional Network, FCN*) для обнаружения объектов и семантической сегментации соответственно. Эти методы концептуально интуитивно понятны и предлагают гибкость и надежность, а также быстрое обучение и время вывода.

Сегментация экземпляров является сложной задачей, поскольку требует правильного обнаружения всех объектов на изображении, а также точной сегментации каждого экземпляра. Таким образом, он сочетает в себе элементы классических задач компьютерного зрения по обнаружению объектов, цель которых состоит в том, чтобы классифицировать отдельные объекты и локализовать каждый с помощью ограничивающей рамки. Учитывая это, можно было ожидать, что для достижения хороших результатов потребуется сложный метод. Однако данные подходы показывают, что удивительно простая, гибкая и быстрая система может превзойти самые современные результаты сегментации экземпляров.

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 масимуму своих возможностей и значимо улучшить их показатели не получалось.

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 ходе процедуры *error analysis* авторы так же разработали метод, позволяющий уменьшить ошибку выделения охватывающей рамки объекта –*bounding-box regression*. После классификации содержимого региона-кандидата, при помощи линейной регрессии на основе признаков из *CNN* определялись четыре параметра — (*dx, dy, dw, dh*). Данные параметры описывали, насколько надо сдвинуть центр рамки региона по *х* и *у*, а также на сколько изменить её ширину и высоту, чтобы точнее охватывать распознанный объект.

Процесс классификации в *SVM* происходит весьма производительно, представляя собой просто матричные операции. Полученные из *CNN* векторы признаков объединяются по всем регионам в матрицу *2000х4096*, которая затем умножается на матрицу *4096xN* с весами *SVM*.

Процедуру детектирования объектов сетью *R-CNN* можно разделить на следующие шаги:

– выделение регионов-кандидатов при помощи *Selective Search*;

– преобразование региона в размер, принимаемый *CNN CaffeNet*;

– получение при помощи *CNN* 4096-размерного вектора признаков;

– проведение *N* бинарных классификаций каждого вектора признаков при помощи *N* линейных *SVM*;

– линейная регрессия параметров рамки региона для более точного охвата объекта.

Общая архитектура сети *R-CNN* изображена на рисунке 2.1.

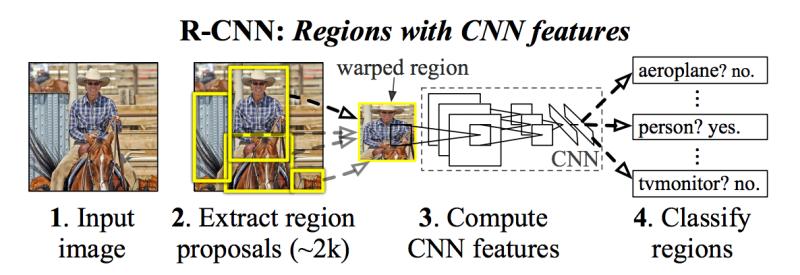


Рисунок 2.1 – Архитектура сети *R-CNN*

По сравнению с классической *R-CNN* сетью, *Fast R-CNN* предлагает 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, предлагает regressor*) совместить все процедуры тренировки в одну.

Общая архитектура *Fast R-CNN* представлена на рисунке 2.2.

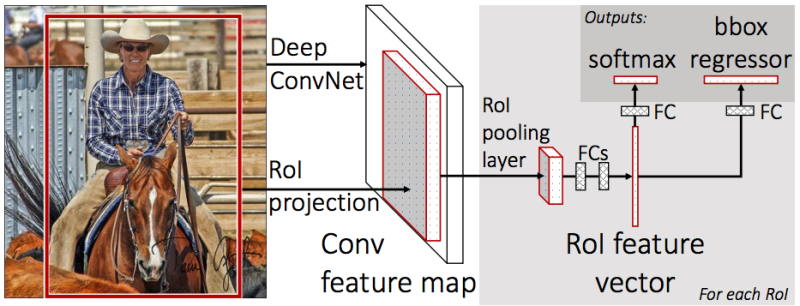


Рисунок 2.2 – Архитектура сети *Fast R-CNN*

Для ускорения вычислений в полносвязанном слое используется разложение матрицы весов по *Truncated SVD*.

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)*. Архитектура сети Faster R-CNN представлена на рисунке 2.3.

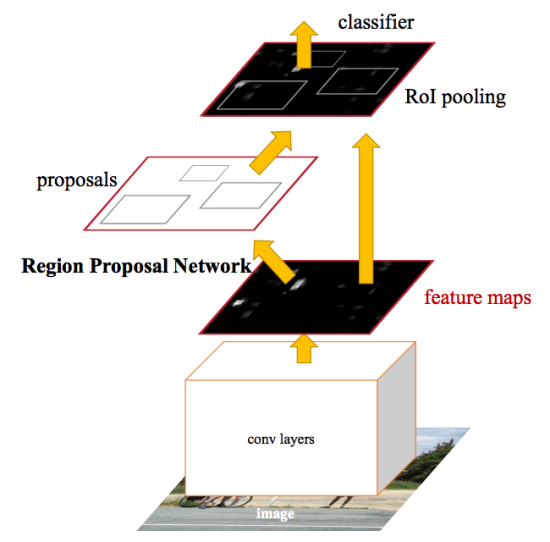


Рисунок 2.3 – Архитектура сети *Faster R-CNN*

В рамках *RPN* по извлечённым *CNN* признакам скользят «мини-нейросетью» с небольшим (3х3) 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* выдаёт по 4 координаты, корректирующие положение охватывающей рамки; *cls*-слой выдаёт по два числа – вероятности того, что рамка содержит хоть какой-то объект или что не содержит.

*Mask R-CNN*, расширяет архитектуру *Faster R-CNN*, добавляя ещё одну ветвь для прогнозирования масок сегментации для каждой интересующей области (*RoI*) параллельно с существующей ветвью для классификации и регрессии ограничивающего прямоугольника, рисунок 2.1.

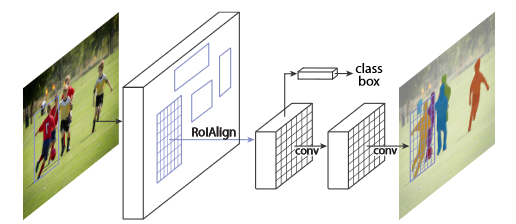


Рисунок 2.1 – Структура *Mask R-CNN* для сегментации изображения

Ветвь маски представляет собой небольшой *FCN*, применяемый к каждому *RoI*, предсказывающий маску сегментации попиксельно. *Mask R-CNN* прост в реализации и обучении, учитывая структуру *Faster R-CNN*, которая обеспечивает широкий спектр гибких архитектурных проектов. Кроме того, ветвь маски добавляет лишь небольшие вычислительные затраты, обеспечивая быструю систему и быстрые эксперименты.

Архитектуру *Mask R-CNN* обычно разделяют на *CNN*-сеть вычисления признаков изображения, называемую *backbone*, и *head* – объединение частей, отвечающих за предсказание охватывающей рамки, классификацию объекта и определение его маски. *Loss* функция для них общая и включает три компонента, формула 1.1.

(1.1)

Выделение маски происходит в *class-agnostic* стиле: маски предсказываются отдельно для каждого класса, без предварительного знания, что изображено в регионе, и потом просто выбирается маска класса, победившего в независимом классификаторе. Такой подход более эффективен, чем опирающийся на априорное знание класса.

## **2.3 Архитектура приложения**

Архитектура клиент-сервер также называется «сеть клиент/сервер» или «модель сетевых вычислений», потому что в этой архитектуре все службы и запросы распространяются по сети. Его функциональность похожа на распределенную вычислительную систему, потому что все компоненты выполняют свои задачи независимо друг от друга.

Архитектура клиент-сервер – это архитектура общей компьютерной сети, в которой несколько клиентов (удаленная система) отправляют множество запросов и, наконец, получают услуги с централизованной серверной машины (хост-системы).

Клиентская машина предоставляет удобный интерфейс, который помогает пользователям запускать службы запросов на серверном компьютере и, наконец, отображать ваши результаты в клиентской системе.

Архитектура клиент-сервер состоит из трех частей:

– *front-end* – это часть программного обеспечения, которая взаимодействует с пользователями, даже если они находятся на разных платформах с разными технологиями, любой интерфейсный модуль в архитектуре клиент-сервер предназначен для взаимодействия со всеми существующими устройствами на рынке, этот уровень содержит экраны входа в систему, меню, экраны данных и отчеты, которые предоставляют и принимают информацию от пользователей, например, большинство инструментов и фреймворков разработки позволяют со-задавать одну версию программы, которая работает для ПК, планшетов и телефонов;

– сервер приложений – это сервер, на котором установлены программные модули приложения, он подключается к базе данных (если она необходима и существует) и взаимодействует с пользователями (называемой клиентской частью);

– сервер базы данных (опционален) – этот сервер содержит таблицы, индексы и данные, которыми управляет приложение, здесь выполняются операции поиска и вставки / удаления / обновления.

Вот несколько преимуществ использования клиент-серверной архитектуры:

– архитектура клиент-сервер разделяет оборудование, программное обеспечение и функциональные возможности системы, например, если требуется адаптация программного обеспечения в конкретной стране, то есть необходимо изменение функциональности, его можно адаптировать в системе без необходимости разрабатывать версию для телефонов, планшетов или ноутбуков.

– поскольку архитектура разделяет аппаратное обеспечение, программное обеспечение и функциональные возможности системы, только интерфейс должен быть адаптирован для взаимодействия с различными устройствами.

Также есть несколько ключевых недостатков:

– если все клиенты одновременно запрашивают данные с сервера, он может быть перегружен.

– если сервер выйдет из строя по какой-либо причине, пользователь не сможет использовать систему.

Для предоставления пользователю сервиса по поиску объектов, человека и ноутбука, на изображении средствами нейронной сети *Mask R-CNN*, *HTTP*-сервер взаимодействует с модулем, в котором хранится вычислительная модель нейронной сети. Взаимодействие с *HTTP*-сервером осуществляет мобильное приложение на платформах *IOS* и *Android*, которая по средствам *HTTP*-протокола отправляет изображение для последующей его обработке.

Так как данное приложение использует *HTTP*-протокол для взаимодействия между клиентом и сервером, оно является в перспективе масштабируемым и расширяемым.

## **2.4 Серверная часть приложения**

Серверная часть приложения реализована благодаря модульной архитектуре, где модули производят взаимодействие между собой. Каждый модуль решает определённую поставленную ему задачу.

В данному случае реализовано два модуля:

– первый модуль решает задачу детектирования объектов на изображении благодаря нейронной сети *Mask R-CNN*, загружая предобученную модель в память компьютера, используя средства, предоставляемые библиотекой *PyTorch*. Входные данные для нейронной сети является изображение в формате *JPEG*, которое благодаря библиотеки *OpenCV* преобразуется в многомерный массив, который фактически и подаётся на входной слой нейронной сети. Результатом нейронной сети является массив данных, который содержит координаты рамок детектированных объектов относительно края изображения;

– второй модуль обрабатывает *HTTP*-запросы клиентов для детектирования изображения. Данный модуль взаимодействует с модулем отвечающий за детектирование изображения, путём извлечения изображения, отправленного пользователем из *HTTP*-запроса, подачи его на вход нейронной сети, и отправки результата детектирования изображения с за ранее выделенными объектами на изображении.

Данные, передаваемые клиентом серверу, имеют формат «Multipart/form-data», это составной тип содержимого, чаще всего использующийся для отправки HTML-форм с бинарными (не-ASCII) данными методом POST протокола HTTP. Указывается в поле заголовка Content-Type (тип содержимого) и следует правилам для составных MIME-данных в соответствии с RFC 2045.

Далее с помощью встроенных средств языка программирования Python, происходит парсинг HTTP-запроса и извлечение из него бинарного изображения, после чего временно заносится в память компьютера для его обработки.

Затем происходит взаимодействие с модулем отвечающий за детектирование объектов на изображении. После чего формируется *HTTP*-ответ, который содержит изображение в формате *base64*.

## **2.5 Клиентская часть приложения**

Клиентская часть приложения реализована с помощью технологии *React Native*, которая позволяет создавать кроссплатформенные мобильные приложения с единым интерфейсом, что позволяет увеличить аудиторию приложения и упростить разработку.

На клиентской части приложения реализован слой доступа к данным, который позволяет производить взаимодействия с сервером. Архитектура слоя доступа к данным изображена на рисунке.

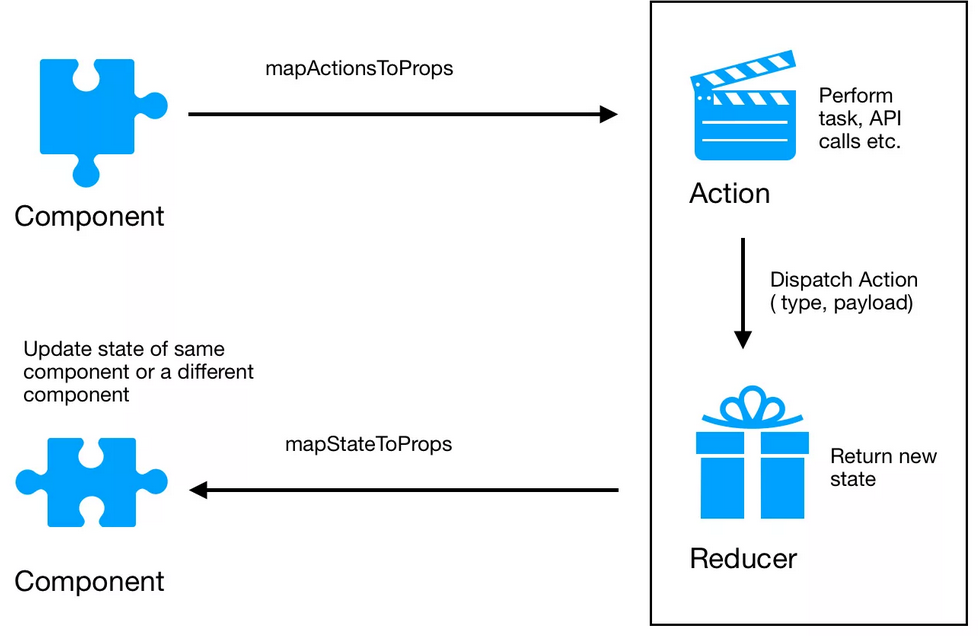


Рисунок 2.2 – Архитектура слоя доступа к данным

Данный слой доступа к данным создаёт для каждого изменяющего объекта состояние, позволяющие менять данные взаимодействуя с сервером динамически, изменяя автоматически графические элементы в интерфейсе приложения, что позволяет абстрагироваться от задачи мониторинга состояния объектов.

Запросы с клиента отравляются с помощью протокола *HTTP*, где формируется *HTTP*-запрос формата «*Multipart/form-data*». Данный *HTTP*-запрос хранит в себе бинарное представление исходного изображения.

Ответ от *HTTP*-сервера хранит в себе изображение в формате *base64*, данный формат декодируется и в последствие выводится изображение с обнаруженными объектами в графическом интерфейсе пользователя.

Данный подход взаимодействие клиента и сервера, позволяет тяжёлые вычислительные операции производить на сервере, при этом разгружая клиента от излишних вычислений, возвращая только результат вычисления.

# 3  РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ КЛАССИФИКАЦИИ ОБЪЕКТОВ

## **3.1** **Средства разработки программного обеспечения**

Для разработки использовался язык программирования *Python*. *Python* – это интерпретируемый объектно-ориентированный язык программирования высокого уровня с динамической семантикой. Его высокоуровневые встроенные структуры данных в сочетании с динамической типизацией и динамическим связыванием делают его очень привлекательным для быстрой разработки приложений, а также для использования в качестве языка сценариев или связующего языка для соединения существующих компонентов. Простой, легкий в освоении синтаксис *Python* подчеркивает удобочитаемость и, следовательно, снижает стоимость обслуживания программы. *Python* поддерживает модули и пакеты, что способствует модульности программы и повторному использованию кода. Интерпретатор *Python* и обширная стандартная библиотека доступны в исходной или двоичной форме бесплатно для всех основных платформ и могут свободно распространяться [4].

Для удобства работы с массивами данных, в *Python*, в программу добавлена библиотека *NumPy*.

*NumPy* – это фундаментальный пакет для научных вычислений на *Python*. Это библиотека *Python*, которая предоставляет объект многомерного массива, различные производные объекты (такие как замаскированные массивы и матрицы) и набор процедур для быстрых операций с массивами, включая математические, логические, манипуляции с формами, сортировку, выбор, ввод-вывод., дискретные преобразования Фурье, базовая линейная алгебра, базовые статистические операции, случайное моделирование и многое другое [3].

В основе пакета *NumPy* лежит объект *ndarray*. Он инкапсулирует *n*-мерные массивы однородных типов данных, при этом многие операции выполняются в скомпилированном коде для повышения производительности. Между массивами *NumPy* и стандартными последовательностями *Python* есть несколько важных отличий:

– массивы *NumPy* имеют фиксированный размер при создании, в отличие от списков *Python* (которые могут расти динамически). Изменение размера *ndarray* создаст новый массив и удалит оригинал;

– все элементы в массиве *NumPy* должны иметь один и тот же тип данных и, следовательно, будут иметь одинаковый размер в памяти. Исключение: можно иметь массивы объектов (*Python*, включая *NumPy*), что позволяет создавать массивы элементов разного размера;

– массивы *NumPy* упрощают сложные математические и другие типы операций с большим количеством данных. Как правило, такие операции выполняются более эффективно и с меньшим количеством кода, чем это возможно при использовании встроенных последовательностей *Python*.

– растущее множество научных и математических пакетов на основе *Python* используют массивы *NumPy*; хотя они обычно поддерживают ввод последовательности *Python*, они преобразуют такой ввод в массивы *NumPy* перед обработкой и часто выводят массивы *NumPy*. Другими словами, чтобы эффективно использовать большую часть современного научного/математического программного обеспечения на основе *Python*, недостаточно просто знать, как использовать встроенные типы последовательностей *Python* – также необходимо знать, как использовать массивы *NumPy*.

*PyTorch* – оптимизированная тензорная библиотека для глубокого обучения с использованием графических процессоров и процессоров.

Пакет *torch* содержит структуры данных для многомерных тензоров и определены математические операции над ними. Кроме того, он предоставляет множество утилит для эффективной социализации тензоров и произвольных типов, а также другие полезные утилиты.

У него есть аналог *CUDA*, который позволяет запускать тензорные вычисления на графическом процессоре *NVIDIA* с вычислительными возможностями >= 3.0.

Тензоры не представляют собой чего-либо особенного, просто являясь многомерными массивами. Тензоры *PyTorch* (*Tensors*) похожи на массивы пакета numpy, но дополнительно могут обрабатываться на видеоускорителях. *PyTorch* поддерживает различные типы тензоров.

Так же *PyTorch* содержит следующие модули:

– *autograd* – *PyTorch* использует метод автоматической дифференциации. Производится запись вычислений, произведенных в прямом направлении, затем производится воспроизведение в обратном порядке для вычисления градиентов. Этот метод особенно полезен при построении нейронных сетей, так как позволяет рассчитывать дифференциальные поправки параметров одновременно с прямым проходом;

– *torch.optim* – модуль, реализующий несколько алгоритмов оптимизации, используемых при построении нейронных сетей. Реализовано большинство наиболее часто используемых методов;

– *torch.nn* – модуль *PyTorch autograd* позволяет легко определять вычислительные графы и работать с градиентами, однако может быть слишком низким уровнем для определения сложных нейронных сетей. Более высокоуровневой абстракцией для таких применений является модуль *nn*.

*PyTorch* предоставляем широкий спектр тензорных, рисунок 3.1, подпрограмм для ускорения и соответствия вашим потребностям научных вычислений, таких как нарезка, индексация, математические операции, линейная алгебра, редукции.

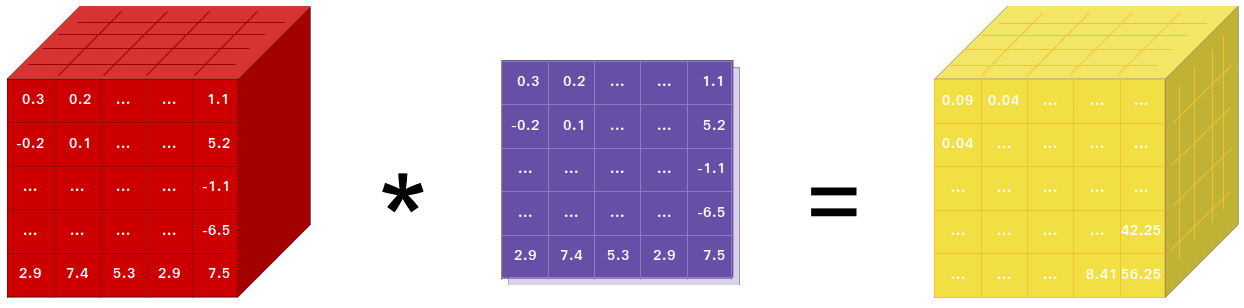


Рисунок 3.1. – Тензоры и операции над ними в *PyTorch*

В *PyTorch* используется техника автоматического дифференцирования в обратном режиме, которая позволяет произвольно изменять способ поведения вашей сети с нулевым запаздыванием или накладными расходами. Данный механизм разработан по нескольким исследовательских работ по этой теме, а также из текущих и прошлых работ, таких как *torch-autograd*, *autograd*, *Chainer* и т.д.

## **3.2 Пользовательский интерфейс приложения**

Пользовательский интерфейс приложения представляет собой кроссплатформенное мобильное приложение, разработанное с помощью технологии *React Native*, взаимодействующие с сервером, на котором реализованная сеть *Mask R-CNN*, которая занимается детектированием на изображении людей и ноутбуков. Взаимодействие клиента с сервером осуществляется с помощью протокола *HTTP*.

Для работы с приложением требуется постоянно работающий *HTTP*-сервер с загруженной в память нейронной сетью. Данный сервер слушает *HTTP* запросы клиентов по внешнему сетевому интерфейсу. Сервер реализует мультипоточную обработку клиентских запросов, следовательно одновременно может обрабатывать сразу несколько клиентских *HTTP*-запросов.

Благодаря модульной архитектуры приложения, нейронная сеть и *HTTP*-сервер является отдельными модулями, которые взаимодействуют между собой, тем самым позволяя в независимости друг от друга расширять возможности определённого модуля.

Для запуска клиентского приложения, требуется запустить соответствующий файл, в зависимости от платформы (*IOS, Android*), на мобильном устройстве. После чего требуется сконфигурировать соединение с сервером, перейдя в соответствующий пункт меню «*Settings*», рисунок 3.2. После чего требуется разблокировать форму ввода, кнопка «*Edit*», и ввести соответствующий порт и *IP*-адрес *HTTP*-сервера, после применения сохранения, кнопка «*Save*», будет успешно производится взаимодействие с сервером.

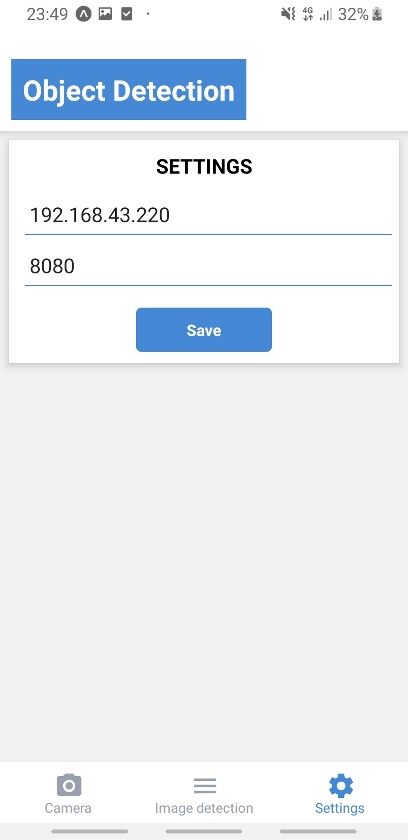
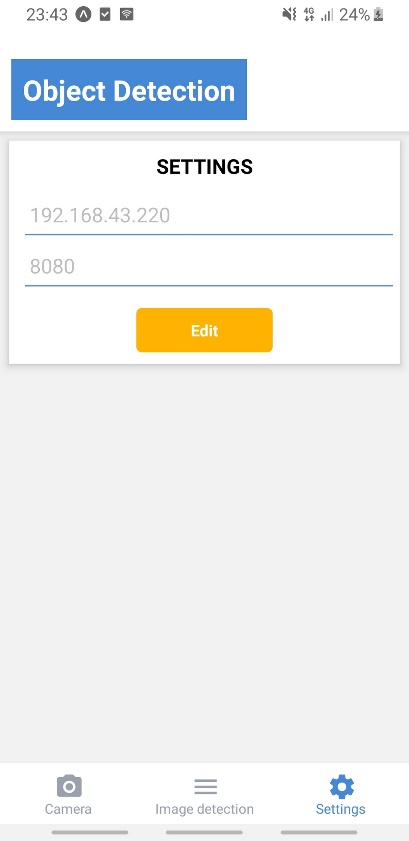


Рисунок 3.2 – Конфигурация клиентского приложения

Получение данных для классификации, производится с помощью камеры смартфона, рисунок 3.3. Для этого требуется выбрать пункт меню «*Camera*», после чего изменится главный экран приложения и откроется доступ к камере.



Рисунок 3.3 – Использование камеры смартфона, для получения объектов для классификации

Далее производится снимок пользователем приложения объектов для классификации, после чего данная фотография сохраняется во временное хранилище на устройстве. После чего данное изображение появляется в пункте меню «*Object detection*», рисунок 3.4.



Рисунок 3.4 – Полученное изображение с объектами для классификации

Для детектирования объектов на изображении, требуется нажать кнопку «*Detect image*», после чего изображение будет отправлено на сервер и произойдёт изменение статуса с «*Raw image*», на «*Processing*». Если соединение с сервером настроено корректно, то с сервера будет получено обработанное изображение с выделенными объектами на изображении, после чего изменится статус на «*Detected image*», рисунок 3.5. Если не удалось соединится с сервером, либо произошла ошибка при передаче данных, статус обратно изменится на «*Raw image*» и будет показано исходное изображение.

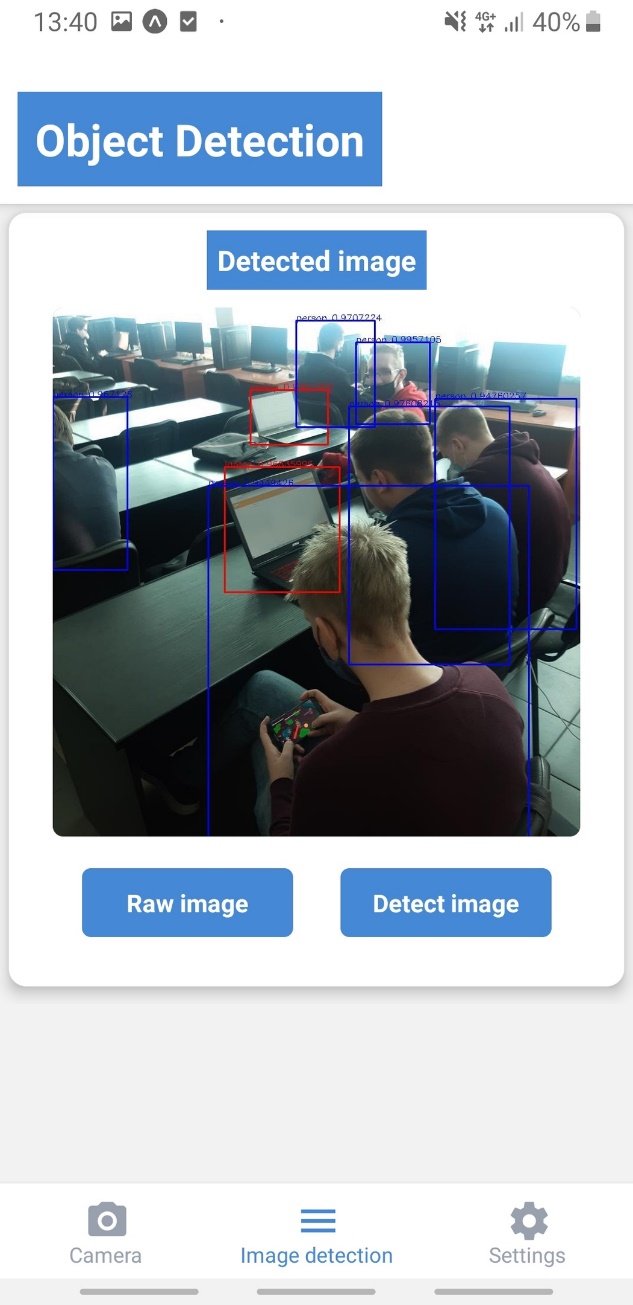


Рисунок 3.5 – Результат детектирования объектов на изображении

Для детектирования объектов на новых данных, требуется произвести повторно действия, описанные выше, после чего будут загружаться в память новые изображение с возможность классификации на них объектов.

## **3.3 Верификация программного обеспечения**

Для верификации приложения, необходимо провести анализ набора изображений из тестовой выборке, и выдать экспертную оценку этих детектирования объектов на изображении.

В качестве тестовой выборки для тестирования приложения, были выбраны 15 случайных изображений разного качества, на которых изображены объекты для детектирования под разными углами и на разном удалении друг от друга.

Предварительно все изображения и объекты на нём были детектированы экспертом, что позволяет произвести верификацию нейронной сети.

Благодаря модульной архитектуре приложения, верфикцию модуля для классификации изображения, можно произвести не затронув модуль для сетевого взаимодействия с клиентом.

После определение тестовой выборки, можно произвести анализ данных с помощью нейронной сети, рисунок 3.6, после чего произвести экспертную оценку результата анализа.

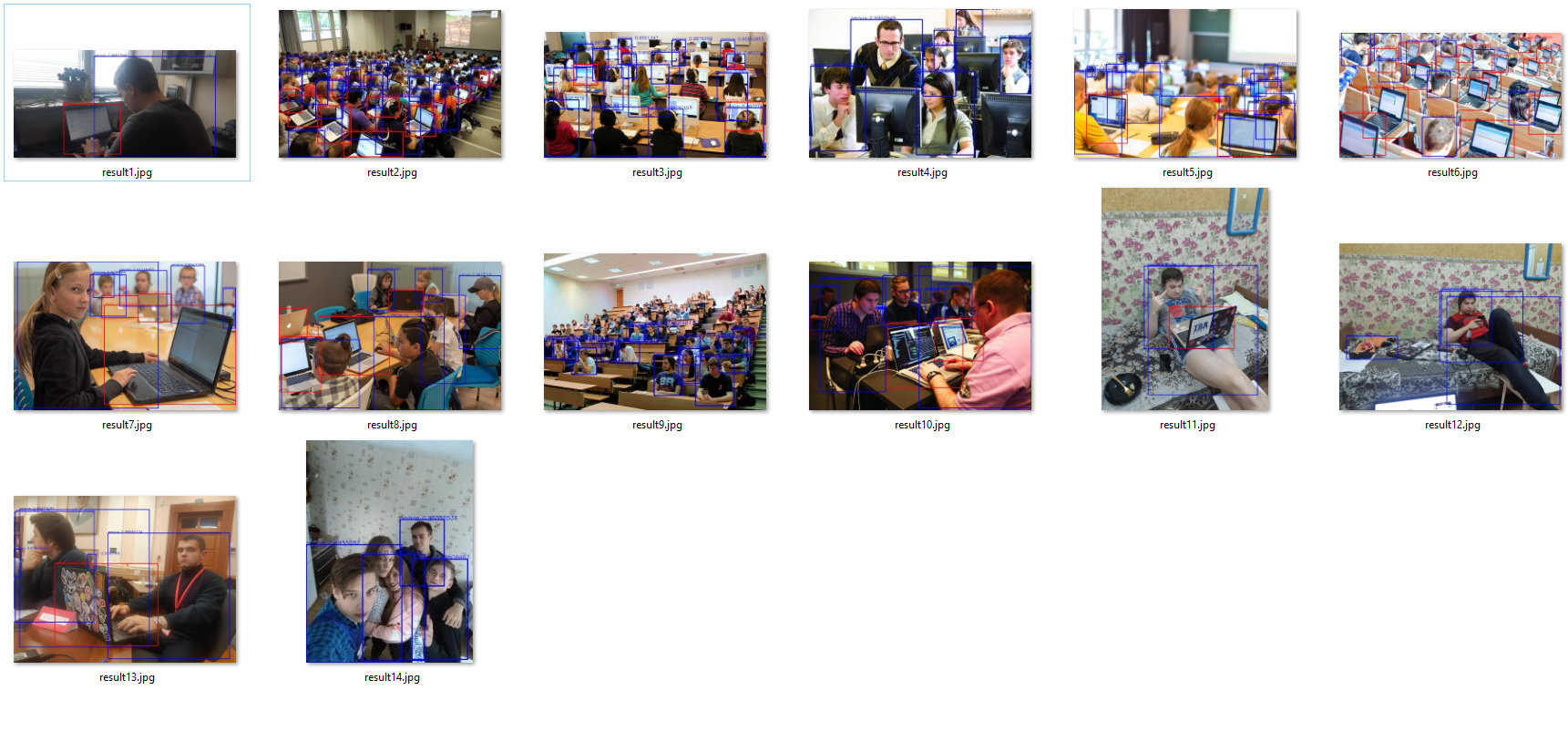


Рисунок 3.6 – Результаты анализа тестовой выборки

По результатам анализа, можно сделать экспертную оценку точности детектирования объектов на изображении.

При попытке произвести классификацию картинок с высоким разрешением (более чем 1500x1500 пикселей), было выбрашенно исключение, связанное с недостатком видеопамяти, данная проблема решается путём оптимизации модели нейронной сети, сжатием изображения до оптимального размера, либо увеличением мощности вычислительного устройства.

Так же на классифицированных изображениях можно заметить, что объекты, которые находятся в далеко, иногда нейронная сеть не может распознать, рисунок 3.7, данная проблема решается, путём увеличения разрешения изображения, либо уточное краёв и цветовой обработки изображения перед анализом, так же можно уменьшить порог классификации объектов, что конечно, может уменьшить точность классификации, из-за чего последует увеличение шума и повысит вероятность ошибочной классификации объектов. В данной задачи эта проблема является не существенной.

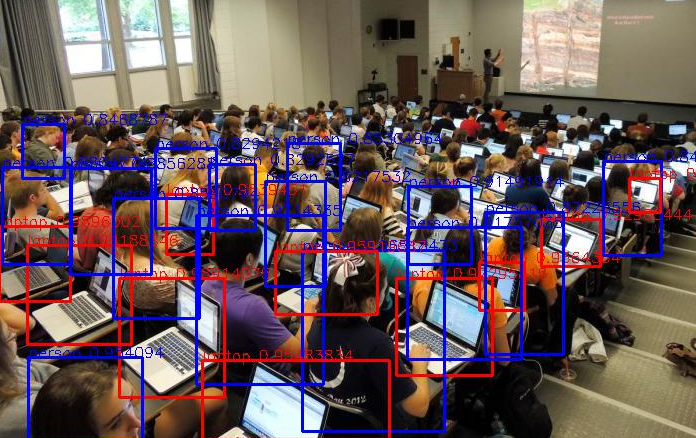
****

Рисунок 3.7 – Результат детектирования изображения на тестовой выборке

При классификации объектов на изображениях, могут встречаться повторные классификации одного и того же объекта, это может происходить из-за необычного ракурса, рисунок 3.8, либо плохого качества изображения. Данную проблему можно решить увелечением порога для принятия решения (например увеличения значения порога до 0.95 отсеет большинство объектов, которые были классифицированы повторно), но данное решение может уменьшить количество детектированных объектов на изображении, если объект находится в далеко или немного размыты его границы.

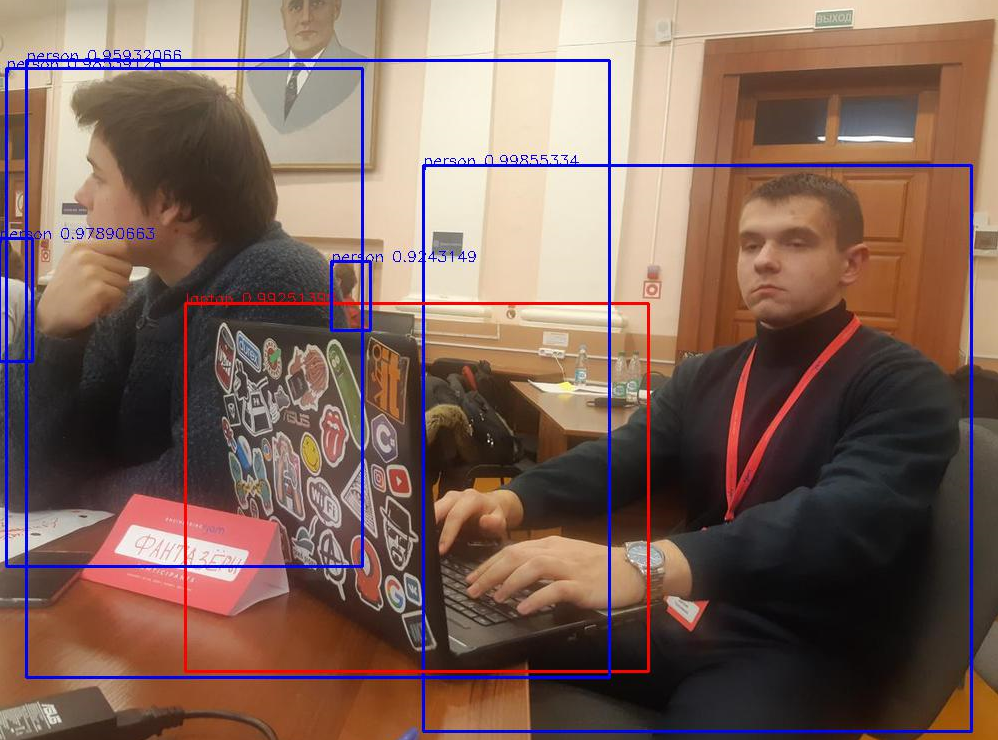


Рисунок 3.8 – Результат детектирования изображения на тестовой выборке

На рисунке 3.9 изображено изображения, в котором имеются размытость на фоне, из-за которой некоторые объекты для детектирования с трудом различимы. После анализа данного изображения, можно убедится в довольно высокой точности определения объектов на изображении (правда не без определённой погрешности), так-как даже размытые объекты, находящиеся на заднем плане, нейронной сети удалось классифицировать.

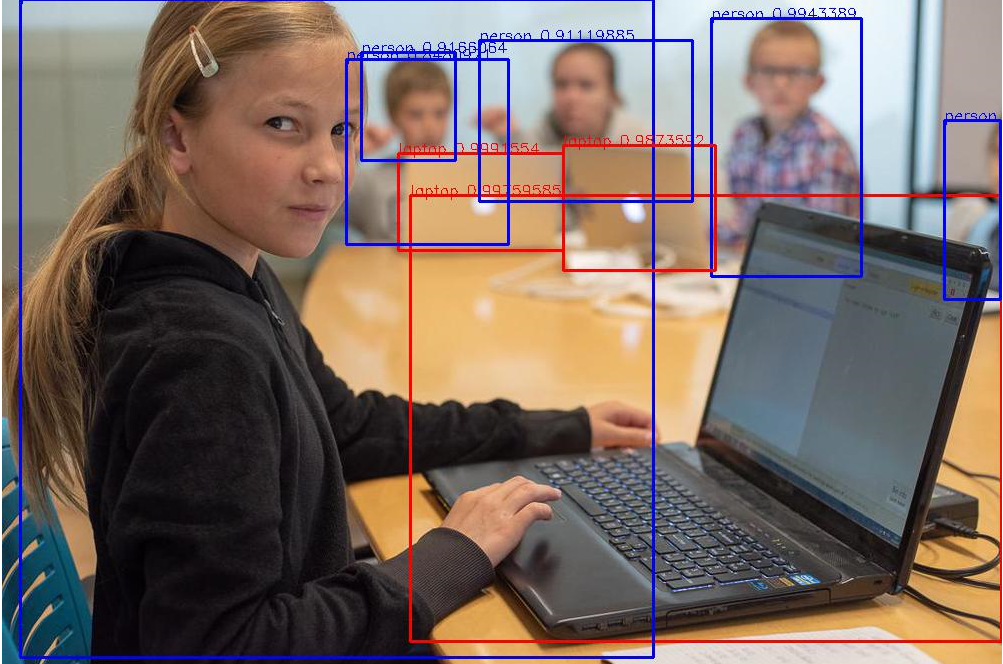


Рисунок 3.9 – Результат детектирования изображения на тестовой выборке

После проведения верификации модуля, отвечающий за классификацию объектов на изображении, можно убедится в том, что определение объектов происходит с достаточной точности, для решения поставленной задачи.

# ЗАКЛЮЧЕНИЕ

Данное программное обеспечение производит детектирование объектов на изображении с достаточной точностью, что в последствии может помочь производить сбор данных, или потоковую обработку данных, для последующего анализа, либо получения определённой информации из изображения.

Так как модуль для классификации изображения определён на сервере и взаимодействие с ним производится через сеть. То данное приложение легко масштабируемо в дальньшем и при его эксплуатации не перегружает клиента излишними вычислениями.

Благодаря проведённой верификации, можно определить точно определения объектов на изображение и с учётом повышения требований к результату, производить оптимизацию средств для классификации объектов.

Разработанный сервис, определения объектов на изображении, использует свои правила обмена информацией, что позволяет ограничиться только необходимой информацией при обмене данными.

Любое клиентское приложение на любой платформе, способной выходить в интернет, может получить доступ к сервису посредством *HTTP* взаимодействия и определить на картинке котов и собак.

Преимущество подобного сервиса заключается в простоте сопровождения и обновления, каждому пользователю нет необходимости скачивать все библиотеки и средства для работы этого приложения, достаточно только простой клиентской части, всё расчёты и обновления происходят только на сервере, который предоставляет доступ к данному сервису.

# Список используемых источников

1. Практическое руководство к курсовому проектированию по курсу «Информатика» для студентов технических специальностей дневной и заочной форм обучения – Гомель: ГГТУ им. П.О. Сухого, 2019. – 32 с.
2. Борисов Е.С. Детектор объектов для неподвижных камер: Сборник трудов конференции. – Севастополь, издательство ФГАОУВО «Севастопольский государственный университет», 2015. – 263 с.
3. Kaiming H., Mask R-CNN Facebook AI Research / Kaiming H., Gkioxari G. – arXiv.org, 2017. – 12 с.
4. Andreas C. Introduction to Machine Learning with Python: A Guide for Data Scientists / C. Andreas, S. Guido. – Paperback, 2017. – 274 с.
5. Бовырин А.В., Разработка мультимедийных приложений с использованием библиотек OpenCV и IPP / П.Н. Дружков, – ИНТУИТ, 2016. – 168 с.
6. Goodfellow I. Deep Learning (Adaptive Computation and Machine Learning series) / I. Goodfellow, Y. Bengio, A. Courville. – Hardcover, 2018. – 1092 с.

# ПРИЛОЖЕНИЕ А

(обязательное)

**Функциональная схема приложения**

# ПРИЛОЖЕНИЕ Б

(обязательное)

**Код программы**

main.py

from PIL import Image

import matplotlib.pyplot as plt

import cv2

import numpy as np

import random

import torch

import torchvision

import torchvision.transforms as T

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

print(device)

model = torchvision.models.detection.maskrcnn\_resnet50\_fpn(pretrained=True).to(device)

model.eval()

CATEGORIES\_IDX = [1, 73]

CATEGORIES = ["person", "laptop"]

def get\_prediction(img\_path, threshold=0.92):

img = T.Compose([T.ToTensor()])(Image.open(img\_path))

pred = model([img.to(device)])

pred\_score = list(pred[0]["scores"].cpu().detach().numpy())

labels = list(pred[0]["labels"].cpu().detach().numpy())

boxes = list(pred[0]["boxes"].cpu().detach().numpy())

masks = list(pred[0]["masks"].cpu().detach().numpy())

pred\_t = [pred\_score.index(x) for x in pred\_score if x > threshold][-1]

masks = (pred[0]['masks'] > threshold).squeeze().detach().cpu().numpy()

masks = masks[:pred\_t + 1]

pred\_class = []

pred\_boxes = []

pred\_masks = []

for i, (label, box, mask) in enumerate(zip(labels, boxes, masks)):

if pred\_score[i] > threshold and label in CATEGORIES\_IDX:

pred\_class.append(CATEGORIES[CATEGORIES\_IDX.index(label)])

pred\_boxes.append([(box[0], box[1]), (box[2], box[3])])

pred\_masks.append(mask)

return pred\_boxes, pred\_class, pred\_score, masks

def random\_colour\_masks(image):

colours = [[0, 255, 0], [0, 0, 255], [255, 0, 0], [0, 255, 255], [255, 255, 0], [255, 0, 255], [80, 70, 180],

[250, 80, 190], [245, 145, 50], [70, 150, 250], [50, 190, 190]]

r = np.zeros\_like(image).astype(np.uint8)

g = np.zeros\_like(image).astype(np.uint8)

b = np.zeros\_like(image).astype(np.uint8)

r[image == 1], g[image == 1], b[image == 1] = colours[random.randrange(0, 10)]

coloured\_mask = np.stack([r, g, b], axis=2)

return coloured\_mask

def detect\_object\_on\_image(img\_path):

boxes, classes, scores, masks = get\_prediction(img\_path)

img = cv2.cvtColor(cv2.imread(img\_path), cv2.COLOR\_BGR2RGB)

for cls, box, score, mask in zip(classes, boxes, scores, masks):

main\_color = (0, 0, 255)

if cls == 'laptop':

main\_color = (255, 0, 0)

cv2.rectangle(img, box[0], box[1], color=main\_color, thickness=2)

cv2.putText(img, cls + " " + str(score), box[0], cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, main\_color, thickness=1)

pil\_img = Image.fromarray(img)

pil\_img.save('tmp/tmp\_detected\_img.jpg')

if \_\_name\_\_ == '\_\_main\_\_':

detect\_object\_on\_image(r"samples/sample10.jpg")

http\_sever.py

import cgi

from http.server import HTTPServer, BaseHTTPRequestHandler, SimpleHTTPRequestHandler

import base64

from PIL import Image

import multipart

from main import detect\_object\_on\_image

from streaming\_form\_data import StreamingFormDataParser

from streaming\_form\_data.targets import ValueTarget, FileTarget, NullTarget

import multipart as mp

# from multipart import to\_bytes

from PIL import Image

from io import BytesIO

class HTTPHandlers(SimpleHTTPRequestHandler):

def do\_POST(self):

print(self.headers)

fields = {}

files = {}

def on\_field(field):

fields[field.field\_name] = field.value

def on\_file(file):

files[file.field\_name] = {'name': file.file\_name, 'file\_object': file.file\_object}

multipart\_headers = {'Content-Type': self.headers.get('Content-Type'),

'Content-Length': self.headers.get('Content-Length')}

multipart.parse\_form(multipart\_headers, self.rfile, on\_field, on\_file)

img = Image.open(BytesIO(fields[None]))

img.thumbnail((1200, 1200), Image.ANTIALIAS)

img.save('tmp/tmp\_img.jpg')

detect\_object\_on\_image('tmp/tmp\_img.jpg')

with open("tmp/tmp\_detected\_img.jpg", "rb") as image\_file:

encoded\_string = base64.b64encode(image\_file.read())

self.send\_response(200)

self.send\_header('content-type', 'text/html')

self.end\_headers()

self.wfile.write(encoded\_string)

if \_\_name\_\_ == '\_\_main\_\_':

httpd = HTTPServer(('', 8080), HTTPHandlers)

print('http server start')

httpd.serve\_forever()

App.js

import \* as React from 'react';

import {StyleSheet, Text, View} from 'react-native';

import {NavigationContainer} from '@react-navigation/native';

import {createMaterialBottomTabNavigator} from '@react-navigation/material-bottom-tabs';

import {StatusBar} from "expo-status-bar";

import {AppHeader} from "./src/components/AppHeader";

import {CameraScreen} from "./src/components/CameraScreen";

import MaterialCommunityIcons from 'react-native-vector-icons/MaterialCommunityIcons';

import GalleryScreen from "./src/components/GalleryScreen";

import Constants from "./src/Constants";

import {StorageState} from "./src/context/storage/StorageState";

import SettingsScreen from "./src/components/SettingsScreen";

const Tab = createMaterialBottomTabNavigator();

export default function App() {

return (

<StorageState>

<NavigationContainer>

<StatusBar style="auto" />

<AppHeader title="Object Detection"/>

<Tab.Navigator

backBehavior='initialRoute'

activeColor={Constants.Colors.MAIN\_COLOR\_BLUE}

style={{ backgroundColor: 'tomato' }}

inactiveColor={Constants.Colors.HINT\_TEXT\_COLOR\_GRAY}

barStyle={{ backgroundColor: Constants.Colors.BACKGROUND\_COLOR\_WHITE }}

>

<Tab.Screen

name="Camera"

component={CameraScreen}

options={{

tabBarLabel: 'Camera',

tabBarIcon: ({ color }) => (

<MaterialCommunityIcons name="camera" color={color} size={26} />

),

}}

/>

<Tab.Screen

name="Image detection"

component={GalleryScreen}

options={{

tabBarLabel: 'Image detection',

tabBarIcon: ({ color }) => (

<MaterialCommunityIcons name="menu" color={color} size={26} />

),

}}

/>

<Tab.Screen

name="Settings"

component={SettingsScreen}

options={{

tabBarLabel: 'Settings',

tabBarIcon: ({ color }) => (

<MaterialCommunityIcons name="settings" color={color} size={26} />

),

}}

/>

</Tab.Navigator>

</NavigationContainer>

</StorageState>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

backgroundColor: 'whitesmoke'

}

});

package.json

{

"main": "node\_modules/expo/AppEntry.js",

"scripts": {

"start": "expo start",

"android": "expo start --android",

"ios": "expo start --ios",

"web": "expo start --web",

"eject": "expo eject"

},

"dependencies": {

"@expo/vector-icons": "^10.2.1",

"@react-navigation/material-bottom-tabs": "^5.3.9",

"@react-navigation/native": "^5.8.9",

"expo": "~39.0.2",

"expo-camera": "~9.0.0",

"expo-media-library": "^9.2.1",

"expo-status-bar": "~1.0.2",

"react": "16.13.1",

"react-dom": "16.13.1",

"react-native": "https://github.com/expo/react-native/archive/sdk-39.0.4.tar.gz",

"react-native-camera": "^3.40.0",

"react-native-paper": "^4.4.0",

"react-native-vector-icons": "^7.1.0"

},

"devDependencies": {

"@babel/core": "~7.9.0"

},

"private": true

}

Constants.js

export default {

Colors: {

MAIN\_COLOR\_BLUE: '#4588d6',

HINT\_TEXT\_COLOR\_GRAY: '#9aa1ae',

BACKGROUND\_COLOR\_WHITE: '#fff',

BLACK: '#000000',

WHITE: '#fff',

RED: '#f00',

ORANGE: '#ffb300'

}

}

AppStyle.js

import Constants from "./Constants";

export default {

Shadows: {

LIGHT\_SHADOW: {

shadowColor: "#000",

shadowOffset: {

width: 0,

height: 3,

},

shadowOpacity: 0.27,

shadowRadius: 4.65,

elevation: 6

}

},

Inputs: {

INPUT\_BORDER\_BOTTOM: {

width: '100%',

fontSize: 18,

padding: 4,

margin: 4,

borderBottomColor: Constants.Colors.MAIN\_COLOR\_BLUE,

borderBottomWidth: 1

}

}

}

SettingsScreenjs

import React, {useContext, useState} from 'react'

import {Text, View, Image, StyleSheet, ScrollView, TouchableOpacity, TextInput, Alert} from 'react-native';

import Constants from "../Constants";

import AppStyles from "../AppStyles";

import {StorageContext} from "../context/storage/storageContext";

import CustomButton from "./utils/CustomButton";

export default () => {

const {serverAddress, serverPort, setServerAddressAndPort} = useContext(StorageContext)

const [editable, setEditable] = useState(false)

const [serverAddressValue, setServerAddressValue] = useState(serverAddress)

const [serverPortValue, setServerPortValue] = useState(serverPort + '')

return (

<View style={styles.container}>

<Text style={styles.title}>SETTINGS</Text>

<View>

<TextInput

style={{...AppStyles.Inputs.INPUT\_BORDER\_BOTTOM}}

value={serverAddressValue}

editable={editable}

onChangeText={setServerAddressValue}

/>

<TextInput

style={{...AppStyles.Inputs.INPUT\_BORDER\_BOTTOM}}

value={serverPortValue}

keyboardType='numeric'

editable={editable}

onChangeText={setServerPortValue}

/>

<View style={styles.controlBar}>

<CustomButton

style={{backgroundColor: editable ? Constants.Colors.MAIN\_COLOR\_BLUE : Constants.Colors.ORANGE}}

content={editable ? 'Save' : 'Edit'}

onPress={() => {

if (!editable) {

setEditable(true)

return ;

}

setServerAddressAndPort(serverAddressValue, serverPortValue)

Alert.alert('Saved' ,'Data successfully saved!')

setEditable(false)

}} />

</View>

</View>

</View>

)

}

const styles = StyleSheet.create({

container: {

backgroundColor: Constants.Colors.WHITE,

padding: 10,

margin: 8,

...AppStyles.Shadows.LIGHT\_SHADOW

},

controlBar: {

paddingTop: 15,

alignItems: 'center'

},

title: {

textAlign: 'center',

fontWeight: 'bold',

fontSize: 18,

paddingBottom: 8

}

})

GalleryScreen,js

import React, {useContext} from 'react'

import {Text, View, Image, StyleSheet, ScrollView} from 'react-native';

import {StorageContext} from "../context/storage/storageContext";

import Constants from "../Constants";

import AppStyles from "../AppStyles";

import CustomButton from "./utils/CustomButton";

import Types from "../context/Types";

const statusDict = {

[Types.STATUS.PROCESSING]: 'Processing..',

[Types.STATUS.RAW\_IMAGE]: 'Raw image',

[Types.STATUS.DETECTED\_IMAGE]: 'Detected image',

}

const colorStatusDict = {

[Types.STATUS.PROCESSING]: Constants.Colors.ORANGE,

[Types.STATUS.RAW\_IMAGE]: Constants.Colors.RED,

[Types.STATUS.DETECTED\_IMAGE]: Constants.Colors.MAIN\_COLOR\_BLUE,

}

const ImageContainer = ({image}) => {

const {detectedImage, showRawImage} = useContext(StorageContext)

return (

<View style={styles.imgContainer}>

<Text style={{...styles.statusText, backgroundColor: colorStatusDict[image.status]}}>

{statusDict[image.status]}

</Text>

<Image style={styles.logo} source={{

uri: image.status === Types.STATUS.DETECTED\_IMAGE ? `data:image/gif;base64,${image.detectedImage.uri}` : image.rawImage.uri

}} />

<View style={styles.controlBtnBox}>

<CustomButton content='Raw image' onPress={() => showRawImage(image)} />

<CustomButton content='Detect image' onPress={() => detectedImage(image)} />

</View>

</View>

)

}

export default () => {

const {images} = useContext(StorageContext)

return (

<ScrollView style={styles.container}>

{images.map(image => <ImageContainer key={image.uriId} image={image}/>)}

</ScrollView>

)

}

const styles = StyleSheet.create({

container: {

flex: 1

},

statusText: {

color: Constants.Colors.WHITE,

padding: 6,

marginBottom: 10,

fontSize: 16,

fontWeight: 'bold'

},

imgContainer: {

flex: 1,

backgroundColor: Constants.Colors.WHITE,

padding: 10,

justifyContent: 'center',

alignItems: 'center',

borderRadius: 10,

margin: 5,

marginBottom: 10,

...AppStyles.Shadows.LIGHT\_SHADOW

},

tinyLogo: {

width: 50,

height: 50,

},

logo: {

width: 300,

height: 300,

backgroundColor: Constants.Colors.HINT\_TEXT\_COLOR\_GRAY,

borderRadius: 6

},

controlBtn: {

backgroundColor: Constants.Colors.MAIN\_COLOR\_BLUE,

color: Constants.Colors.WHITE,

fontWeight: 'bold',

borderRadius: 5,

padding: 10,

textAlign: 'center',

width: 120

},

controlBtnBox: {

flexDirection: 'row',

justifyContent: 'space-around',

width: '100%',

padding: 18

}

});

Footer.js

import React from 'react'

import {View, Button, StyleSheet} from "react-native";

export const Footer = () => (

<View style={styles.footer}>

<Button color='#4a76a8' title="Take photo"/>

<Button color='#4a76a8' title="Processed images"/>

<Button color='#4a76a8' title="Setting"/>

</View>

)

const styles = StyleSheet.create({

footer: {

flex: .1,

flexDirection: 'row',

justifyContent: 'space-around',

alignItems: 'center',

backgroundColor: 'white',

shadowColor: "#000",

shadowOffset: {

width: 0,

height: 3,

},

shadowOpacity: 0.27,

shadowRadius: 4.65,

elevation: 6

},

button: {

width: '50%',

backgroundColor: '#f00'

}

})

CustomCamera.js

import React, {useContext, useEffect, useState} from 'react'

import {Alert, StyleSheet, Text, TouchableOpacity, View, Vibration} from "react-native";

import {Camera} from 'expo-camera';

import Icon from "react-native-vector-icons/MaterialCommunityIcons";

import Constants from "../Constants";

import {StorageContext} from "../context/storage/storageContext";

const CameraUtilIconBtn = ({name = 'refresh', onPress, style, iconSize = 56, color = 'white'}) => {

return (

<TouchableOpacity

style={{

...styles.cameraUtilIconBtn,

...style

}}

onPress={onPress}>

<Icon name={name} size={iconSize} color={color} />

</TouchableOpacity>

)

}

const SnapshotBtn = ({onPress, radius}) => (

<TouchableOpacity style={{alignSelf: 'center'}} onPress={onPress}>

<View style={{

borderWidth: 2,

borderRadius: 50,

borderColor: Constants.Colors.WHITE,

height: radius,

width: radius,

display: 'flex',

justifyContent: 'center',

alignItems: 'center'}}

>

<View style={{

borderWidth: 2,

borderRadius: 50,

borderColor: Constants.Colors.WHITE,

height: radius - 10,

width: radius - 10,

backgroundColor: Constants.Colors.WHITE}} >

</View>

</View>

</TouchableOpacity>

)

export default () => {

const [hasPermission, setHasPermission] = useState(null);

const [cameraRef, setCameraRef] = useState(null)

const [type, setType] = useState(Camera.Constants.Type.back);

const [flashMode, setFlashMode] = useState(Camera.Constants.FlashMode.off);

const {addRawImage} = useContext(StorageContext)

useEffect(() => {

(async () => {

const { status } = await Camera.requestPermissionsAsync();

setHasPermission(status === 'granted');

})();

}, []);

if (hasPermission === null) {

return <View />;

}

if (hasPermission === false) {

return <Text>No access to camera</Text>;

}

return (

<Camera style={{flex: 1}} type={type} flashMode={flashMode} onMountError={() => null} ref={ref => {

setCameraRef(ref) ;

}}>

<View

style={{

flex: 1,

backgroundColor: 'transparent',

flexDirection: 'row',

alignItems: 'flex-end',

}}>

<View style={{paddingBottom: 20, flexDirection: 'row', width: '100%', justifyContent: 'space-around'}}>

<CameraUtilIconBtn name='refresh' onPress={() => {

setType(

type === Camera.Constants.Type.back

? Camera.Constants.Type.front

: Camera.Constants.Type.back

);

}} />

<SnapshotBtn

onPress={async() => {

if(cameraRef){

let photo = await cameraRef.takePictureAsync();

Vibration.vibrate(100)

addRawImage(photo)

// console.log('photo', photo);

}

}}

radius={80}

/>

<CameraUtilIconBtn

name='flash'

onPress={() => setFlashMode(

flashMode === Camera.Constants.FlashMode.off

? Camera.Constants.FlashMode.torch

: Camera.Constants.FlashMode.off

)}

/>

</View>

</View>

</Camera>

)

}

const styles = StyleSheet.create({

cameraUtilIconBtn: {

alignSelf: 'flex-end',

alignItems: 'center'

},

snapshotBtn: {

backgroundColor: Constants.Colors.BLACK,

padding: 0.1,

borderRadius: 80,

borderColor: Constants.Colors.BACKGROUND\_COLOR\_WHITE,

borderWidth: 2

}

})

CameraScreen.js

import React, {useEffect, useState} from 'react'

import {StyleSheet, View} from "react-native";

import CustomCamera from "./CustomCamera";

export const CameraScreen = ({navigation}) => {

const [cameraLoad, setCameraLoad] = useState(true)

// TODO: Need optimize

useEffect(() => {

navigation.addListener('tabPress', e => {

setCameraLoad(true)

});

navigation.addListener('blur', e => {

setCameraLoad(false)

});

}, [navigation])

return (

<View style={{flex: 1}}>

{cameraLoad &&

<CustomCamera/> }

</View>

)

}

AppHeader.js

import React from 'react'

import {Text, View, StyleSheet} from "react-native";

import AppStyles from "../AppStyles";

import Constants from "../Constants";

export const AppHeader = ({title}) => (

<View style={styles.header}>

<Text style={styles.text}>{title}</Text>

</View>

)

const styles = StyleSheet.create({

header: {

flex: .19,

alignItems: 'flex-start',

justifyContent: 'flex-end',

backgroundColor: 'white',

// height: 90,

...AppStyles.Shadows.LIGHT\_SHADOW

},

text: {

fontSize: 25,

backgroundColor: Constants.Colors.MAIN\_COLOR\_BLUE,

fontWeight: 'bold',

color: 'white',

padding: 10,

margin: 10

}

})