МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

РАБОТА ПРОВЕРЕНА

Рецензент
Доцент кафедры информатики,
информационных технологий и
методики обучения информатике
ФГБОУ ВО «ЮУрГГПУ», к.п.н.,
доцент

_____ О.А. Дмитриева

«___»_____ 2024 г.

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Разработка нейросетевой модели детектирования и отслеживания электросамокатов в транспортном потоке города

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.04.04.2024.308-1494.ВКР

Научный руководитель,
доцент кафедры СП, к.п.н.
_____ О.Н. Иванова

Автор работы,
студент группы КЭ-228
_____ Д.И. Мякотин

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы магистранта
студенту группы КЭ-228
Мякотину Дмитрию Игоревичу,
обучающемуся по направлению
09.04.04 «Программная инженерия»
(магистерская программа «Искусственный интеллект и инженерия данных»)

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)
Разработка нейросетевой модели детектирования и отслеживания электросамокатов в транспортном потоке города.
2. **Срок сдачи студентом законченной работы:** 20.05.2024 г.
3. **Исходные данные к работе**

3.1. Gawande U., Hajari K., Golhar Y. Real-time deep learning approach for pedestrian detection and suspicious activity recognition. // Procedia Computer Science. 2023. – Т. 218 – С. 2438–2447.

3.2. Li C., Wang Y., Liu X. An improved YOLOv7 lightweight detection algorithm for obscured pedestrians. // Sensors (Basel), 2023. – Т. 23, № 13. – С. 5912–5926.

4. **Перечень подлежащих разработке вопросов**

4.1. Провести обзор научной литературы.

4.2. Разработать нейросетевую модель.

4.3. Провести обучение и тестирование нейросетевой модели.

4.4. Спроектировать и разработать веб-приложение.

4.5. Протестировать веб-приложение.

4.6. Провести анализ полученных результатов.

5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
доцент кафедры СП, к.п.н. О.Н. Иванова

Задание принял к исполнению Д.И. Мякотин

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

В современном мире использование электросамокатов стало очень популярным, однако это вызывает обеспокоенность по поводу безопасности других участников дорожного движения. Детектирование и отслеживание водителей электросамокатов в транспортном потоке является важной задачей. Это позволит в будущем увеличить безопасность на дорогах и тротуарах.

Постановка задачи

Целью выпускной квалификационной работы является разработка нейросетевой модели детектирования и отслеживания электросамокатов в транспортном потоке города.

Для достижения поставленной цели необходимо решить следующие задачи:

1) провести обзор научной литературы и существующих приложений по предметной области;
2) собрать и разметить набор данных;
3) разработать нейросетевую модель;
4) провести обучение и тестирование нейросетевой модели;
5) спроектировать и разработать веб-приложения;
6) провести тестирование веб-приложения;
7) провести анализ полученных результатов.

Структура и содержание работы

Работа состоит из введения, пяти глав, заключения и списка литературы. Объем работы составляет 58 страниц, объем списка литературы – 25 источников.

В первой главе было проведено описание предметной области, а именно описание методов машинного обучения и нейронной сети, а также задача классификации. Помимо вышеперечисленного был проведен обзор

научной литературы и аналогов и описание семейство нейросетевых моделей YOLOv7.

Во второй главе была описана архитектура нейросетевой модели YOLOv7, ее основные части и их назначение. Был описан набор данных для обучения и тестирования модели, описано веб-приложение «RoboFlow» в котором выполнялась разметка изображений. Также был описан алгоритм Deep SORT для отслеживания объектов в реальном времени.

В третьей главе были сформулированы функциональные и нефункциональные требования к разрабатываемому веб-приложению, спроектирована диаграмма вариантов использования, которая описывает взаимодействия пользователя с веб-приложением для детектирования и трекинга электросамокатов.

В четвертой главе была реализована нейросетевая модель для детектирования и трекинга электросамокатов в транспортном потоке. Модель была обучена на собранном наборе данных. Также было реализовано веб-приложение для демонстрации работы нейросетевой модели. Веб-приложение полностью соответствует функциональным и нефункциональным требованиям.

В пятой главе было проведено тестирование нейронной сети, тестирование проводилось на разных моделях YOLOv7, наилучший результат показала модель YOLOv7-w6. Также было проведено функциональное тестирование веб-приложения. Было проведено автоматизированное тестирование веб-приложения с помощью инструмента cypress, веб-приложение было протестировано с помощью двух видов тестирования, компонентного и E2E.

В приложении А содержится код для детектирования и трекинга.

В приложении Б содержится код серверной части веб-приложения.

В приложении В содержится код клиентской части веб-приложения.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Изучение предметной области

Подход, при котором прошлые данные или примеры используются для первоначального формирования и совершенствования схемы предсказания, называется методом машинного обучения (Machine Learning) [1].

Машинное обучение – чрезвычайно широкая и динамически развивающаяся область исследований, использующая огромное число теоретических и практических методов.

Выделяются два способа машинного обучения. При первом способе часть совокупности данных – обучающая выборка – выделяется только для обучения. После того как метод предсказания определяется по обучающей выборке, более он не изменяется и в дальнейшем используется для решения задачи предсказания. При втором способе обучение никогда не прекращается, т.е. оно происходит в режиме онлайн, тем самым предсказания и обучение происходят постоянно в процессе поступления новых данных [1].

Одним из алгоритмов машинного обучения является нейронная сеть. Нейронная сеть содержит узлы – аналоги нервных клеток – нейронов (нейроподобных элементов, НПЭ) и их соединения – синапсические связи. Каждый нейрон управляем или извне, или по принципу самоуправления с помощью обратных связей. Можно регулировать значения весов синапсических связей и значения порогов. Такое регулирование, реализовано в разных моделях, и определяет возможность обучения и самообучения сети. Оно задает пути прохождения возбуждений через сеть, простейшим образом формируя связи «посылка – следствие». Обучение заканчивается тогда, когда вероятность «узнавания» достигнет требуемого значения, т.е. необходимость корректировки параметров по предъявляемым эталонам возникает все реже [2].

Задача классификации изображений

Распознавание изображений – это процесс обработки входного изображения нейронной сетью и присвоении метки для этого изображения или объекту на изображении.

Задача классификации – распределение некоторого множества объектов по заданному множеству групп (классов). При этом есть некоторое подмножество объектов, для которых распределение по классам известно, классовая принадлежность остальных – неизвестна [3].

1.2. Обзор научной литературы

В статье [4] авторы решали проблему пропуска пешеходов в сценах движения с высокой плотностью пешеходов при детектировании. Для решения данной проблемы они использовали современный алгоритм YOLOv7 в качестве базового алгоритма детектирования. На основе данного алгоритма авторы предлагают облегченную сеть обнаружения пешеходов под название HRECSA-NET, которая решает распространенную проблему окклюзии пешеходов в сценах с высокой плотностью. Данный алгоритм способен лучше справляться с задачами обнаружения заслоненных и маленьких пешеходов. Авторы использовали легкую сверточную нейронную сеть MobilenetV3 для замены backbone (слой, который отвечает за начальную подготовку данных для дальнейшей работы нейронной сети) модели YOLOv7, что уменьшило размер модели до 1/6 от первоначального размера, тем самым значительно ускорился вывод. Кроме этого, разработчики создали пирамидную сеть высокого разрешения, чтобы помочь сети извлекать данные из видимых областей заслоненных пешеходов и пешеходов небольшого размера, что позволяет уменьшить частоту пропусков таких пешеходов. И последнее, но не менее важное, – это встраивание в слой head (слой, который отвечает за классификацию полученных объектов) эффективного модуля ECSA-Net, который позволяет избегать извлечения ненужной информации и фонового

шума, позволяя алгоритму обнаружения отфильтровывать значительное количество избыточных кадров, тем самым сокращая вычисления последующих кадров обнаружения.

В результате авторы получили алгоритм, который способен обнаруживать пешеходов с сильной окклюзией, что значительно повышает точность обнаружения пешеходов в сценах с высокой плотностью. Однако предложенный алгоритм не способен справиться с проблемой обнаружения пешеходов, которые полностью скрыты в течении короткого периода времени.

Точность алгоритма HRECSA-NET на наборе данных CrowdHuman (который содержит фотографии пешеходов с высокой плотностью), составило 89,75% по метрике mAP. Результат работы алгоритма HRECSA-NET представлен на рисунке 1.

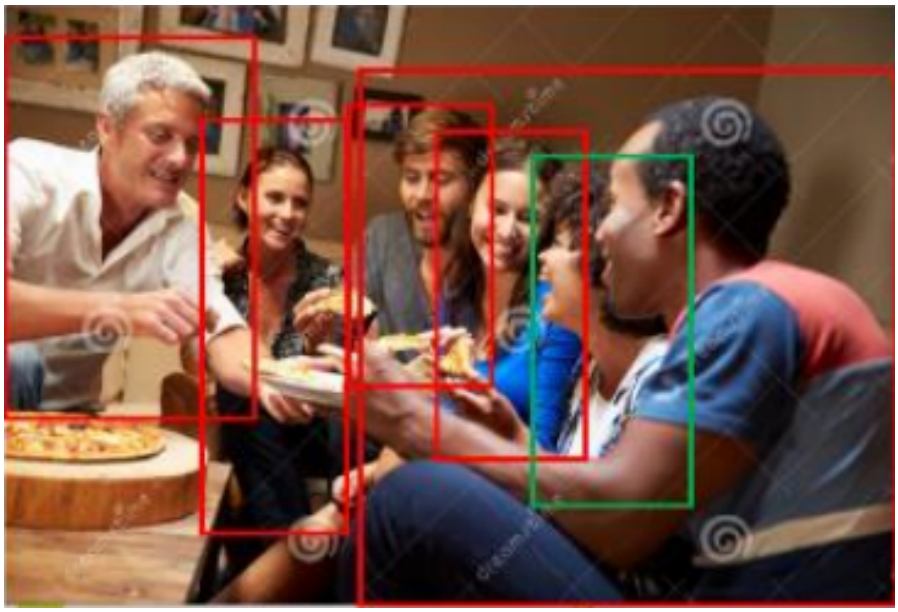

Рисунок 1 – Результат работы алгоритма HRECSA-NET [4]

В статье [5] авторы решают важную задачу восприятия окружающей среды автономных транспортом, с помощью алгоритмов машинного обучения. Поскольку велосипедисты являются уязвимыми участниками дорожного движения, обнаружение и отслеживание велосипедистов является важной задачей, для того чтобы избегать столкновения с транспортом.

Для решения данной задачи авторы предлагают надежный метод обнаружения и отслеживания велосипедистов, основанного на многослойном лазерном сканере IBEO LUX 4L, который получает четырехслойное облако точек из локальной среды. Сначала лазерные точки разбиваются на отдельные кластеры с помощью метода пространственной кластеризации на основе плотности (DBSCAN). Затем 37-мерный набор признаков оптимизируется с помощью алгоритма рельефа и анализа главных компонентов (PCA) для получения двух двумерных кластеров. Метод опорных векторов (SVM) и алгоритм «Дерево решения» комбинируются с тремя наборами признаков. Также алгоритм отслеживания множественных гипотез (MHT) и фильтр Калмана применяются для отслеживания движущихся велосипедистов и оценки их движения.

По словам авторов, алгоритм протестирован в реальных условиях и показывает хорошие результаты. В будущем авторы планируют использовать различные датчики для достижения более точного обнаружения и отслеживания велосипедистов на большом расстоянии.

На рисунке 2 представлен лазерный луч, распределяющийся на четыре слоя, и сканер, установленный на транспортное средство.

Рисунок 2 – Лазерный луч и сканер на транспортном средстве [5]

В статье [6] авторы предлагают метод генерации ограничительной рамки на основе сегментации, для всенаправленного обнаружения пешеходов. Данный алгоритм позволяет детекторам плотно подогнать ограничительную рамку к пешеходу без определенного ракурса. Авторы считают, что благодаря широкому углу зрения, всенаправленные камеры являются более экономичными, чем стандартные камеры, и они подходят для крупномасштабного мониторинга. Однако современные детекторы не могут быть напрямую применены для обнаружения пешеходов на изображении с таких камер, потому что изображение с такой камеры может быть повернуто на любой угол, что существенно ухудшает точность обнаружения. Существующие методы решают данную проблему путем преобразования изображения во время вывода, однако эти методы имеют низкую скорость обнаружения. Есть еще один метод, который позволяет обойтись без преобразования, ценой трудоемкой работы по разметки данных и обучению детекторов с помощью размеченного набора данных.

Для решения данной проблемы авторы предлагают вместо этого использовать существующий крупномасштабный набор данных для обнаружения объектов, чтобы избежать как преобразований, так и разметки набора данных. Они обучают детектор с помощью повернутых изображений и плотно подогнанной разметки, ограничивающей рамок, созданной на основе разметки сегментации в наборе данных, что позволяет детекторам обнаруживать пешеходов на всенаправленных изображениях. Авторы также разработали псевдоискажение «рыбий глаз», который деформирует изображения, чтобы имитировать искажения «рыбий глаз», что больше повышает производительность.

Тестирование данного метода производилось на разных наборах данных результат на наборе данных MW-18Mar достигает 50,6 по метрике AP, на наборе данных HABBOF 46,8 по метрике AP и на наборе данных CEPDOF 33,4 по метрике AP. Результат работы данного метода представлен на рисунке 3.

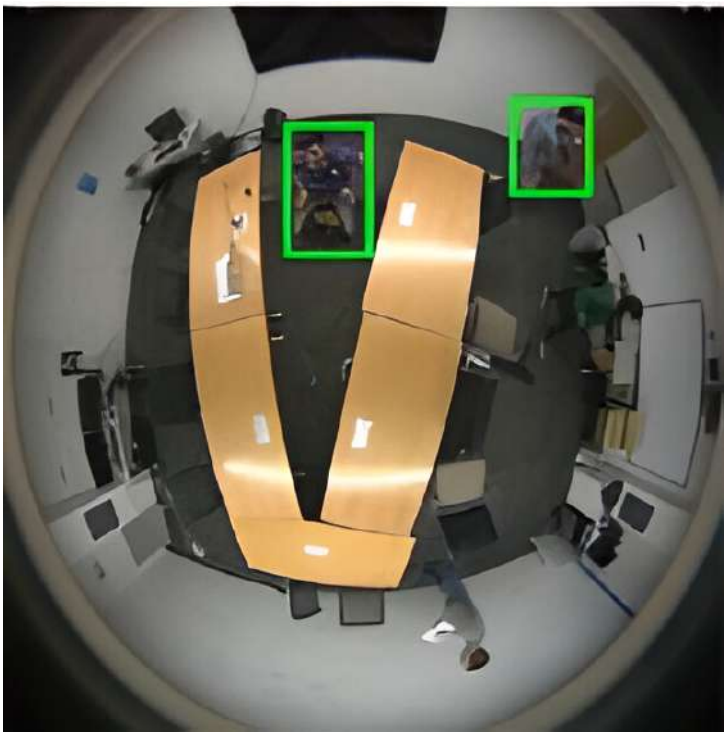

Рисунок 3 – Результат работы метода для всенаправленного обнаружения пешеходов [6]

В статье [7] авторы решают проблему обнаружения и классификации уязвимых участников дорожного движения, ведь это является критически важным требованием безопасности для развертывания автономного транспорта в неоднородном трафике. Хоть по внешнему виду водители электросамокатов и велосипедов схожи, но они имеют совершенного другие характеристики движения и могут развивать скорость до 45 км/ч. Проблема обнаружения водителей электросамокатов усугубляется в городских условиях среды, где очень часто участники движения загораживают друг друга, что может привести к не обнаружению или не правильной классификации водителей электросамокатов. В данном исследовании авторы представили новый метод для обнаружения частично загороженных водителей электросамокатов другими участниками дорожного движения. Данный алгоритм достигает повышения точность до 15,93% для обнаружения частично загороженных водителей электросамокатов по сравнению с другими методами. Авторы говорят, что существует большой простор для будущих улучшений в

данной области. Более крупный и разнообразный наборы данных для обучения смогут повысить эффективность обнаружения, высокопроизводительное оборудование необходимо для достижения требований автономного транспортного средства в режиме реального времени при близком расстоянии, особенно для такого сценария как появления велосипедистов и водителей электросамокатов из-за припаркованных машин или других препятствий. Результат работы данного метода представлен на рисунке 4.

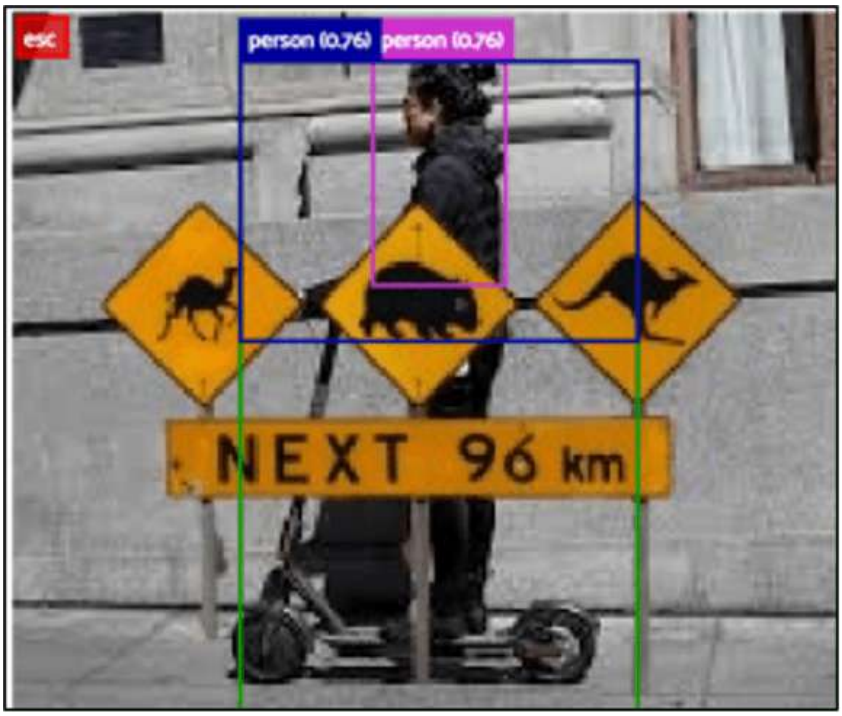

Рисунок 4 – Результат работы метода для обнаружения частично загороженных водителей электросамокатов [7]

В статье [8] авторы решали проблему обнаружения и классификации обычных пешеходов и водителей электросамокатов. Авторы считают, что количество электросамокатов продолжает расти и поскольку поведение водителя электросамоката сильно отличается от поведения обычных пешеходов, это может создать новые проблемы для систем активной безопасности транспортных средств и автономных транспортных средств. Авторы утверждают, что их система на основе компьютерного зрения является одной из

первых в мире. Данная система может детектировать, а также отличать водителей электросамокатов от обычных пешеходов, помимо этого авторы представили свой набор данных для обучения данной системы, который содержит изображения пешеходов и водителей электросамокатов. Система построена на основе двух существующих современных сверточных нейронных сетей, YOLOv3 и MobileNetV2. С помощью данной комбинации авторы смогли добиться очень хороших результатов и точность системы достигает 91%.

Результаты тестирования данной системы по различным метрикам представлены в таблице 1.

Таблица 1 – Результаты тестирования системы [8]

Класс	**Precision**	**Recall**	**F1**
Водитель электросамоката	0,95	0,91	0,93
Обычный пешеход	0,98	0,99	0,98

Примеры работы данной системы и пример изображения из набора данных представлены на рисунке 5 [8].

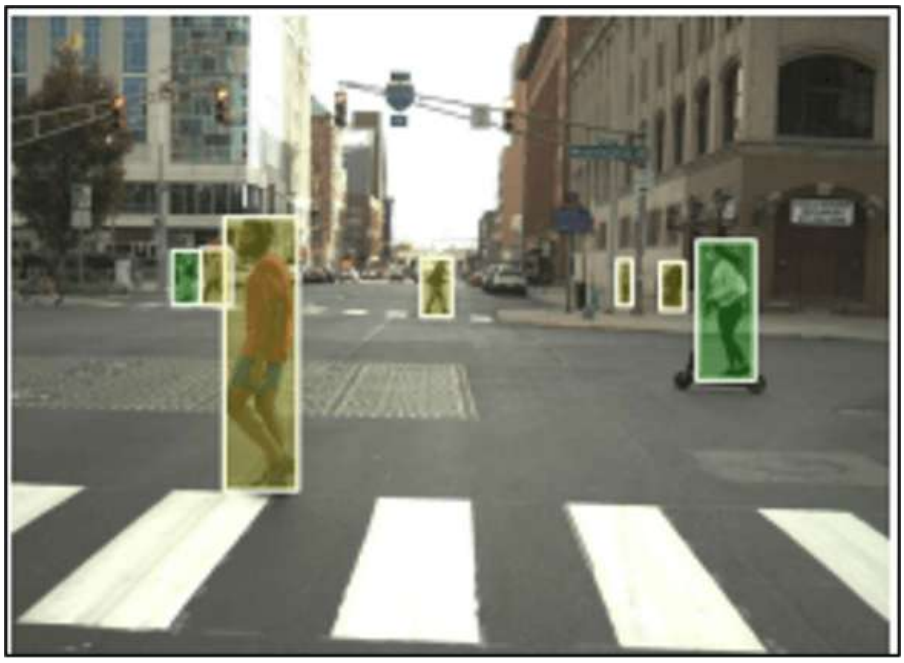

Рисунок 5 – Результаты работы системы, построенной на YOLOv3 и MobileNetV2

В статье [9] авторы говорят о проблеме обнаружения, отслеживания пешеходов и распознавания подозрительных действий пешеходов. Непрерывный мониторинг частных и общественных зон в районах с высокой плотностью населения очень сложен, поэтому требуется активное видеонаблюдение, которое способно отслеживать поведение пешеходов в реальном времени. В данной работе авторы представили инновационную и надежную систему глубокого обучения, а также уникальный набор данных. Помимо набора данных в исследовании предложена улучшенная по масштабу модель Mask R-CNN, которая может эффективно обнаруживать пешеходов и распознавать подозрительные активности. Данная модель является особенно эффективна и экономична по времени обучения и тестирования по сравнению с ее обычной версией. Авторы считают, что данное исследование в будущем поможет добиться еще больше результатов в данной области. Сами авторы планируют в будущем провести дополнительные эксперименты для оценки позы и траектории движения пешеходов.

В статье [10] авторы говорят о проблеме обнаружения объектов на изображениях низкого качества и объектов маленького размера. Для решения данной проблемы в статье предлагается метод обнаружения крошечных объектов TOD-YOLOv7 на основе YOLOv7. Авторы реконструировали сеть YOLOv7 путем добавления слоя обнаружения крошечных объектов для повышения ее способности к обнаружению. Затем они использовали модуль рекурсивной закрытой свертки для того, чтобы ускорить процесс инициализации модели. Также в данной статье предлагается интеграция механизма координированного внимания в сеть извлечения признаков YOLOv7 для лучшего восприятия информации о пешеходах.

Для тестирования данной сети использовался набор данных TinyPerson, который содержит изображения, с людьми, которые находятся на очень большом расстоянии. Результаты показывают, что по сравнению с базовой моделью YOLOv7, точность обнаружения предложенной модели

достигает улучшения от 7,1% до 9,5%, а скорость обучения достигает 208 кадров в секунду (FPS) [10].

Результаты работы данной модели представлены на рисунке 6.

Рисунок 6 – Результаты работы модели TOD-YOLOv7 [10]

1.2. YOLOv7

You Only Look Once (YOLO) – это семейство моделей обнаружения сложных объектов, которые были предварительно обучены на наборе данных COCO. Особенность семейства моделей YOLO заключается в том, что она применяет сверточные нейронные сети на изображение всего один раз в отличии от ее аналогов.

YOLOv7 превосходит все известные детекторы объектов как по скорости, так и по точности в диапазоне от 5 до 160 кадров в секунду и обладает самой высокой точностью 56,8% AP среди всех известных детекторов объектов реального времени с частотой 30 кадров в секунду.

Нейросетевая модель YOLOv7 реализована с помощью PyTorch [11] и имеет 6 версий моделей, а именно YOLOv7, YOLOv7-X, YOLOv7-W6, YOLOv7-E6, YOLOv7-D6, YOLOv7-E6E [12]. Сравнение данных моделей с другими представлено на рисунке 7.

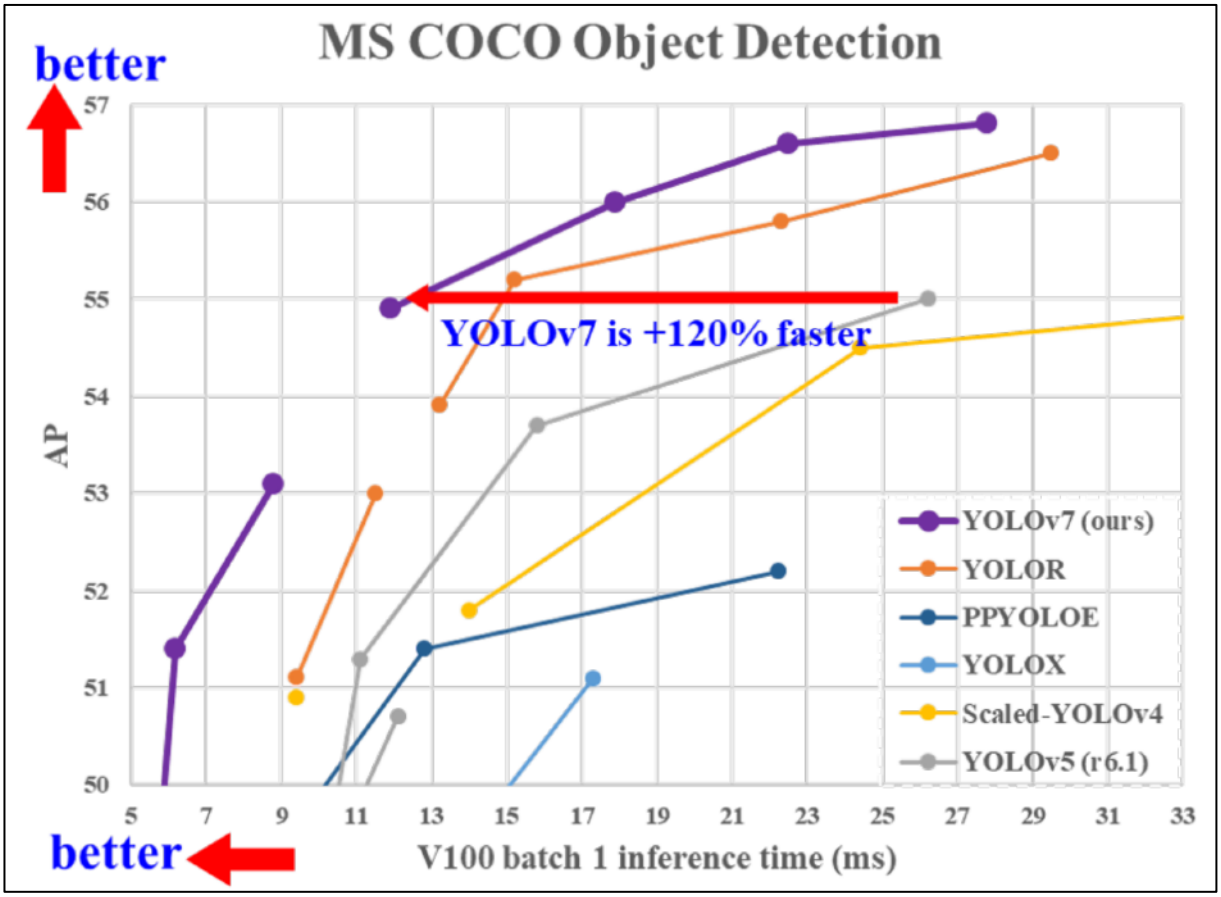

Рисунок 7 – Сравнение моделей YOLOv7 [12]

Подробное описание архитектуры нейросетевой модели YOLOv7 представлено во второй главе.

Вывод по первой главе

В данной главе было проведено описание предметной области, а именно описание машинного обучение и нейронной сети, а также задача классификации. Помимо вышеперечисленного был проведен обзор научной литературы и аналогов, которые были предложены в этих работах, а также было описано семейство нейросетевых моделей YOLOv7.

2. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

2.1. Описание архитектуры нейросетевой модели YOLOv7

Архитектура нейросетевой модели VOLOv7 состоит из 3 основных компонентов backbone, neck и head. Архитектура YOLOv7 представлена на рисунке 8.

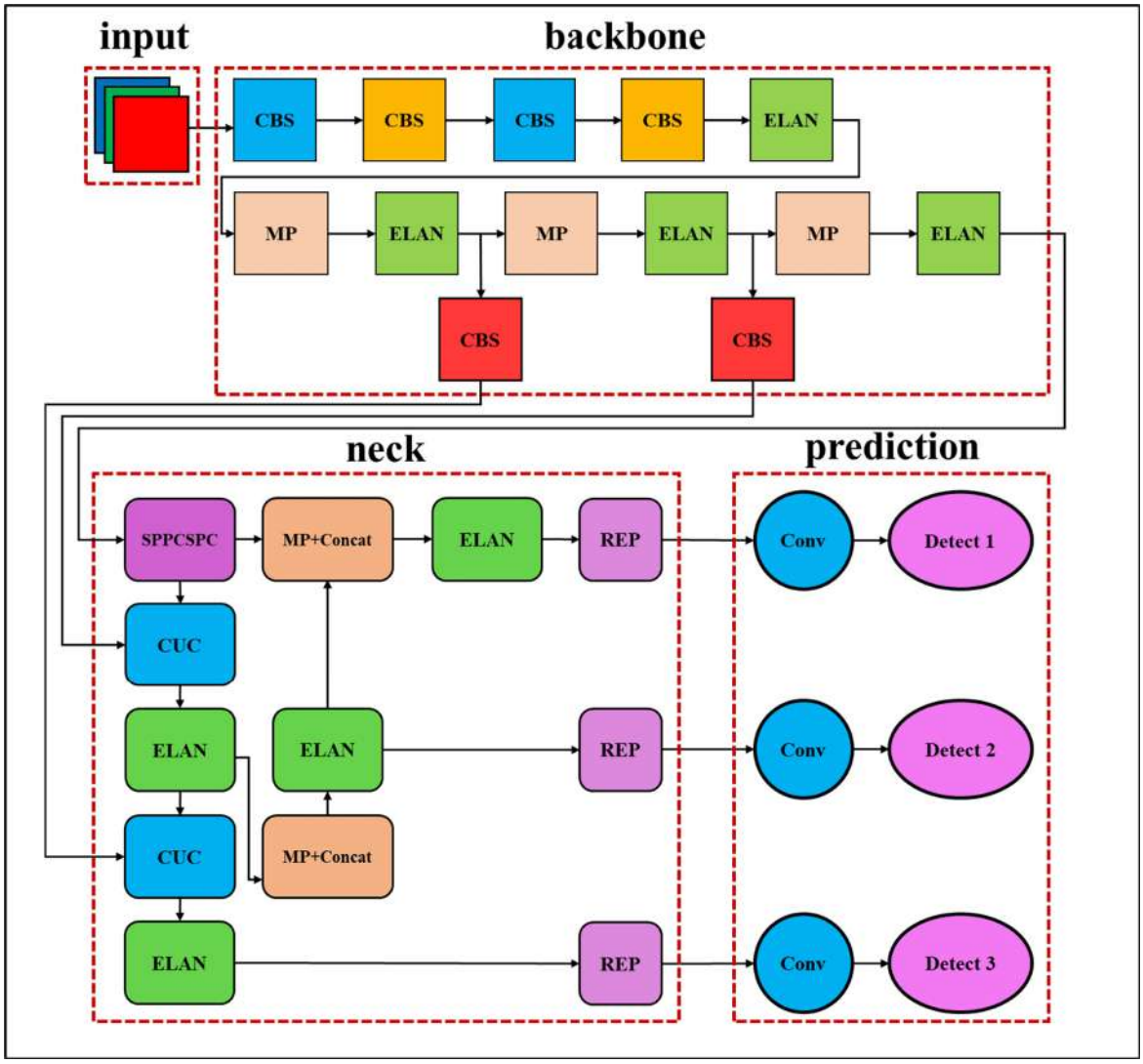

Рисунок 8 – Архитектура нейросетевой модели YOLOv7 [13]

Backbone

Данная часть архитектуры отвечает за начальную подготовку данных для дальнейшей работы нейронной сети. Входные изображения проходят через сверточные слои, которые отвечают за извлечение признаков из изображений, также изображения проходят через слои пулинга которые предназначены для решения проблемы фиксированного размера карты признака.

Это позволяет устранить потери низкоуровневых признаков на раннем этапе.

Neck

Данная часть архитектуры отвечает за объединение карт признаков, полученных из предыдущего слоя. Она использует идею архитектуры PANet (Path Aggregation Network), которая помогает справиться с проблемой потери информации при прохождении изображения через множество слоев нейронной сети. После многократной обработки изображения сложность вычислений возрастает, а разрешение изображения уменьшается, что может привести к утрате важных признаков. Архитектура PANet позволяет оптимизировать вычисления, обеспечивая эффективное объединение признаков снизу-вверх, что сокращает путь и помогает распознавать больше признаков на изображении. Архитектура PANet представлена на рисунке 9.

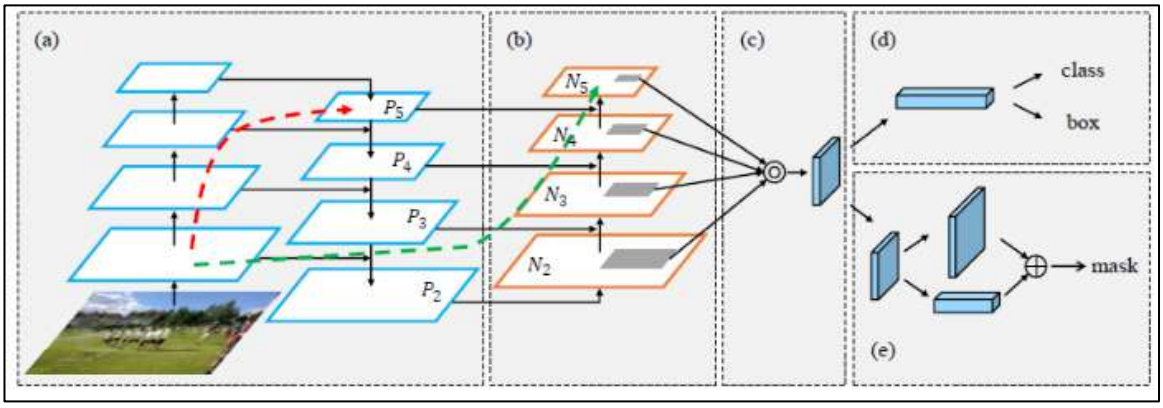

Рисунок 9 – Архитектура сети PANet [13]

Head

Данная часть архитектуры представлена в виде полносвязной нейронной сети, которая состоит из трех слоев. Полносвязная нейронная сеть отвечает за выполнение финальных вычислений и преобразований признаков, необходимых для точного определения местоположения объектов и их классификации на изображении или видео. Этот этап завершает процесс обработки изображений и формирует окончательные выводы модели обнаружения объектов.

2.2.Набор данных

Для обучения и тестирования модели нейронной сети использовался набор данных IUPUI-CSRC E-Scooter Rider [15] который содержит 10 749 изображений водителей электросамокатов и 10 705 изображений не водителей электросамокатов и набор данных Occluded E-Scooter Rider Detection Dataset [16] который содержит изображения загороженных водителей электросамокатов. Примеры изображений из наборов данных представлены на рисунках 10–11.

Рисунок 10 – Пример изображения из набора данных

Рисунок 11 – Второй пример изображения из набора данных

Разметка изображений выполнялась с помощью веб-приложения «RoboFlow» [17]. Данное приложение имеет удобный и понятный интерфейс, а также большое количество полезного функционала для разметки изображений. Приложение позволяет разметить изображение, после чего сохранить их и распределить в процентном соотношении на обучающую, тренировочную и валидационную выборку. После этого можно выбрать необходимый формат и скачать на компьютер. Интерфейс приложения «RoboFlow» представлен на рисунке 12. В результате разметки изображений в данном сервисе был получен набор данных изображений 640x640 с водителями электросамокатов, в количестве 3600 изображений и поделен на обучающую (2520 изображений), валидационную (720 изображений) и тестовую (360 изображений) выборку в соотношении 70/20/10. Пример размеченного изображения представлен на рисунке 13.

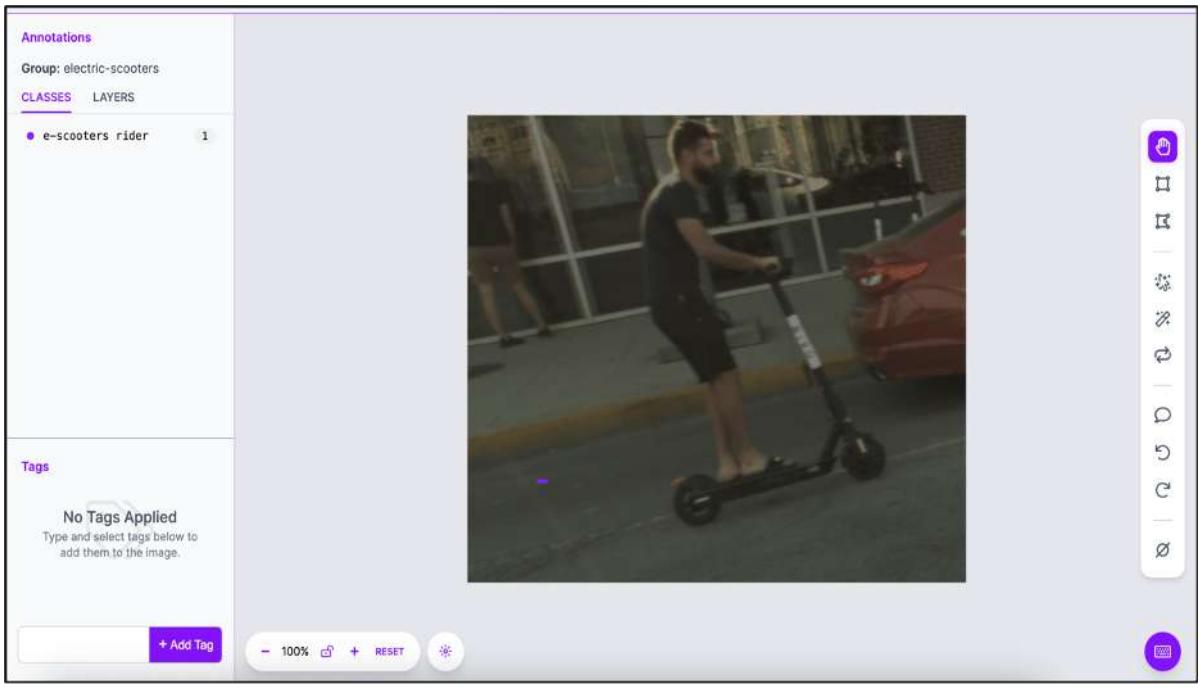

Рисунок 12 – Интерфейс для разметки изображений веб-приложения «RoboFlow»

Рисунок 13 – Пример размеченного изображения

Маркировка YOLO представляет из себя текстовый файл, который содержит аннотацию ограничивающего прямоугольника (Bounding Box, BBox) для каждого изображения. Аннотации нормализуются по размеру изображения и лежат в диапазоне от 0 до 1. Первое число это номеркласса, остальные значение это координаты ограничивающего прямоугольника. Пример текстового файла с разметкой представлен на рисунке 14.

```
0 0.3015625 0.48046875 0.36953125 0.83828125
0 0.6109375 0.375 0.34609375 0.74921875
```

Рисунок 14 – Пример текстового файла с разметкой в формате YOLOv7

2.3. Алгоритм отслеживания

После процесса детектирования объектов с помощью нейросетевой модели YOLOv7 необходимо отслеживать объект. Для решения задачи отслеживания был использован алгоритм глубокой сортировки (Deep SORT algorithm) [18]. Данный алгоритм – это модификация алгоритма SORT (Simple Online and Realtime Tracking) [19]. Алгоритм Deep SORT начинает свою работу с детектирования объекта с помощью нейронный сетей для детектирования таких как семейство моделей YOLO или других. Алгоритм Deep SORT, как и алгоритм SORT содержит в себе фильтр Калмана, который необходим для прогнозирования объекта на основе таких параметров как скорость, позиция и ускорение, благодаря чему фильтр Калмана предсказывает положение объекта в текущем кадре.

Также данный алгоритм использует Венгерский алгоритм и Расстояние Махаланобиса. Венгерский алгоритм связывает идентификатор с обнаруженным объектом, для того чтобы в дальнейшем сопоставлять его с нужным объектом в кадре, это позволяет предсказывать положение объекта если он на короткое время зайдет за преграду. Расстояние Махалонобиса используется для улучшения процесса сопоставления объектов в кадре. В сравнении с евклидовым расстоянием, расстояние Махаланобиса учитывает корреляцию между признаками объекта, что позволяет алгоритму более точно определять сходство между объектами.

Вывод по второй главе

В данной главе была описана архитектура нейросетевой модели YOLOv7, ее основные части и их назначение. Был описан набор данных для обучения и тестирования модели, описано веб-приложение «RoboFlow», в котором выполнялась разметка изображений. Также был описан алгоритм Deep SORT для отслеживания объектов в реальном времени.

3. ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЕ

3.1. Функциональные и нефункциональные требования

Перед непосредственной реализацией приложении необходимо определить и продумать функциональные и нефункциональные требования системы.

Функциональные требования – это требования, которые описывают функционал, сервисы или действия, которые приложения должно выполнять. Они определяют, как должна работать система, какие операции она должна выполнять и как пользователи будут взаимодействовать с ней.

Нефункциональные требования — это требования, которые не связаны с функциональностью системы, они скорее определяют ее качественные характеристики, такие как производительность, безопасность, надежность, удобство и другие качества.

Разрабатываемое приложение должно удовлетворять следующим функциональным требованиям:

1) веб-приложение должно предоставлять множественную возможность загрузки изображений;

2) веб-приложение должно распознавать водителей электросамокатов на загруженных изображений;

3) веб-приложение должно предоставлять возможность скачивания zip архива, в котором будут все загруженные изображения;

4) веб-приложение должно предоставлять возможность просмотра обработанных изображений;

5) веб-приложение должно выполнять отслеживание водителей электросамокатов на видео;

6) веб-приложение должно принимать на вход файлы изображений в формате jpg или png;

7) веб-приложение должно принимать на вход файл видео в формате mp4;

8) веб-приложение должно иметь возможность дозагрузки файлов изображений;

9) веб-приложение должно иметь возможность запускать видео с результатом трекинга;

10) веб-приложение должно иметь возможность остановить видео с результатом трекинга;

11) веб-приложение должно выделять водителей электросамокатов в цветную рамку на загруженных изображений;

12) во время обработки изображений веб-приложение не должно позволять загружать изображения.

Разрабатываемое приложение должно удовлетворять следующим нефункциональным требованиям:

1) клиентская часть веб-приложения должна быть реализована с помощью фреймворка Next.js [20];

2) клиентская часть веб-приложения должна быть реализована с помощью языка программирования TypeScript [21];

3) клиентская часть веб-приложения должна быть реализована с помощью языка разметки HTML;

4) клиентская часть веб-приложения должна быть реализована с помощью языка стилей CSS;

5) клиентская часть веб-приложения должна тестироваться с помощью инструмента для автоматизированного тестирования Cypress [22];

6) серверная часть веб-приложения должна быть реализована с помощью языка программирования Python [23];

7) серверная часть веб-приложения должна быть реализована с помощью фреймворка FastAPI [24];

8) в случае ошибки работы серверной части, должны выводиться ошибки на клиентской части;

9) во время процесса обработки должен отображаться значок обработки.

3.1. Диаграмма вариантов использования приложения

В процессе проектирования системы была построена диаграмма вариантов использования на языке графического описания для объектного моделирования в области разработки программного обеспечения UML (Unified Modeling Language). Диаграмма вариантов использования описывает взаимодействия актера «Пользователя» с веб-приложением для детектирования и трекинга водителей электросамокатов. Она описывает все возможное взаимодействие пользователей с разрабатываемой системой. Диаграмма вариантов использования представлена на рисунке 15.

На диаграмме вариантов использования изображен актер «Пользователь», который взаимодействует с веб-приложением. Ему доступны следующие варианты использования:

1) запустить процесс детектирования водителей электросамокатов на загруженных изображениях;
2) выбрать файлы изображений;
3) перетащить файлы изображений в зону для загрузки;
4) посмотреть результат детектирования на изображениях;
5) выбрать следующие изображение;
6) скачать архив с обработанными изображениями;
7) запустить процесс трекинга водителей электросамокатов на загруженном видео;
8) выбрать файл видео;
9) перетащить файл видео в зону для загрузки;
10) посмотреть результат трекинга на видео;
11) запустить обработанное видео;
12) остановить обработанное видео.

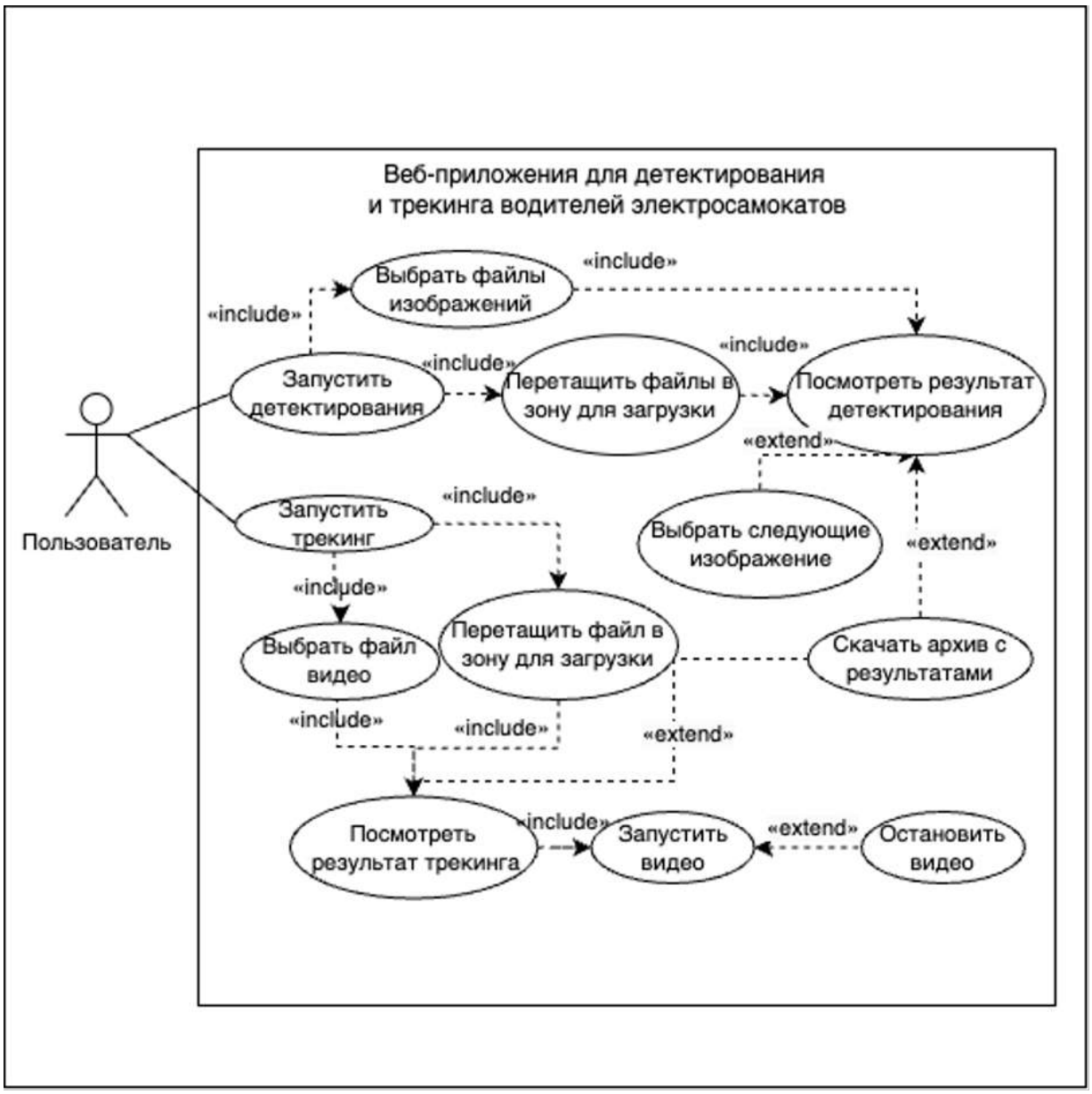

Рисунок 15 – Диаграмма вариантов использования

Вывод по третьей главе

В данной главе были сформулированы функциональные и нефункциональные требования к разрабатываемому веб-приложению, спроектирована диаграмма вариантов использования на языке графического описания для объектного моделирования в области разработки программного обеспечения UML (Unified Modeling Language), которая описывает взаимодействия пользователя с веб-приложением для детектирования и трекинга водителей электросамокатов.

4. РЕАЛИЗАЦИЯ

4.1. Программные средства реализации

Для реализации веб-приложения для детектирования и трекинга водителей электросамокатов на изображении или видео были использованы языки программирования Python, который использовался для реализации нейронной сети и серверной части, и TypeScript, который использовался для реализации клиентской части. Для написания кода использовались IDE PyCharm Community Edition и Visual Studio Code.

Средства реализации нейронной сети представлены ниже.

1. PyTorch – библиотека для машинного обучения с открытым исходным кодом [11].
2. YOLOv7 – нейросетевая модель для распознавания и классификации объектов на изображениях [12].
3. Deep SORT – алгоритм, используемый для отслеживания нескольких объектов на видео [18].

Средства реализации клиентской части приложения представлены ниже.

1. Next.js – это фреймворк для веб-разработки с открытым исходным кодом [20].
2. HTML – стандартный язык разметки для документов, предназначенных для отображения в веб-браузере.
3. CSS – язык стилей, для стилизации веб-страниц.

Для реализации серверной части приложения использовался современный веб-фреймворк, для создания RESTful API на языке Python FastAPI [24].

4.2. Реализация и обучение нейросетевой модели YOLOv7

Для обучения нейросетевой модели был собран набор данных из 3600 изображений и поделен на обучающую (2520 изображений), валидационную (720 изображений) и тестовую (360 изображений) выборку в соотношении 70/20/10.

Обучение YOLOv7 запускается с помощью скрипта train.py. В параметры скрипта необходимо указать количество эпох обучения, количество обучающих примеров за один проход, путь до конфигурационного файла набора данных (data.yaml), данный файл содержит всю информацию о наборе данных, а именно пути для обучающей, валидационной и тестовой выбоки, количество классов и их имена.

Обучение нейронной сети проводилось на облачной платформе Google Colaborator [25] и GPU Nvidia Tesla T4. Обучение проводилось на 50 эпохах, обучение заняло 3 часа. В ходе обучения сохранялись различные графики метрик и функции потерь для обучающей и тестовой выборки набора данных. Графики метрики mAP, recall, precision и функции потерь представлены на рисунках 16–19. На рисунке 16 отображено динамическое изменения метрики mAP на протяжении 50 эпох, наилучший результат показала модель yolov7-w6.

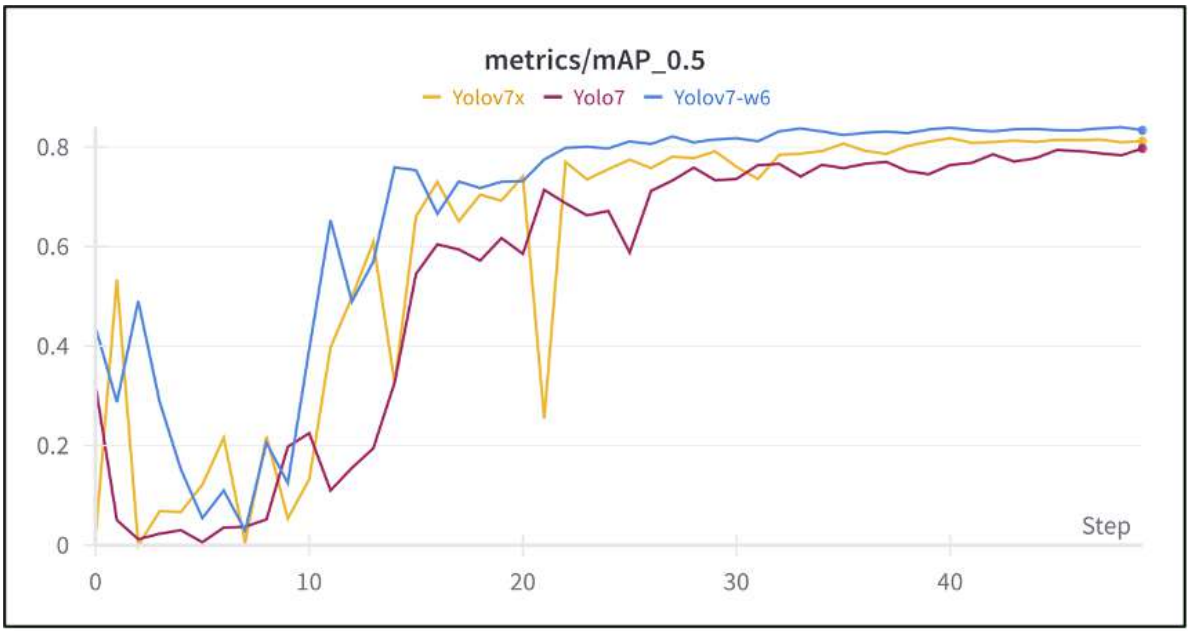

Рисунок 16 – График метрики mAP

На рисунке 17 отображено динамическое изменения метрики precision на протяжении 50 эпох, наилучший результат показала модель yolov7-w6.

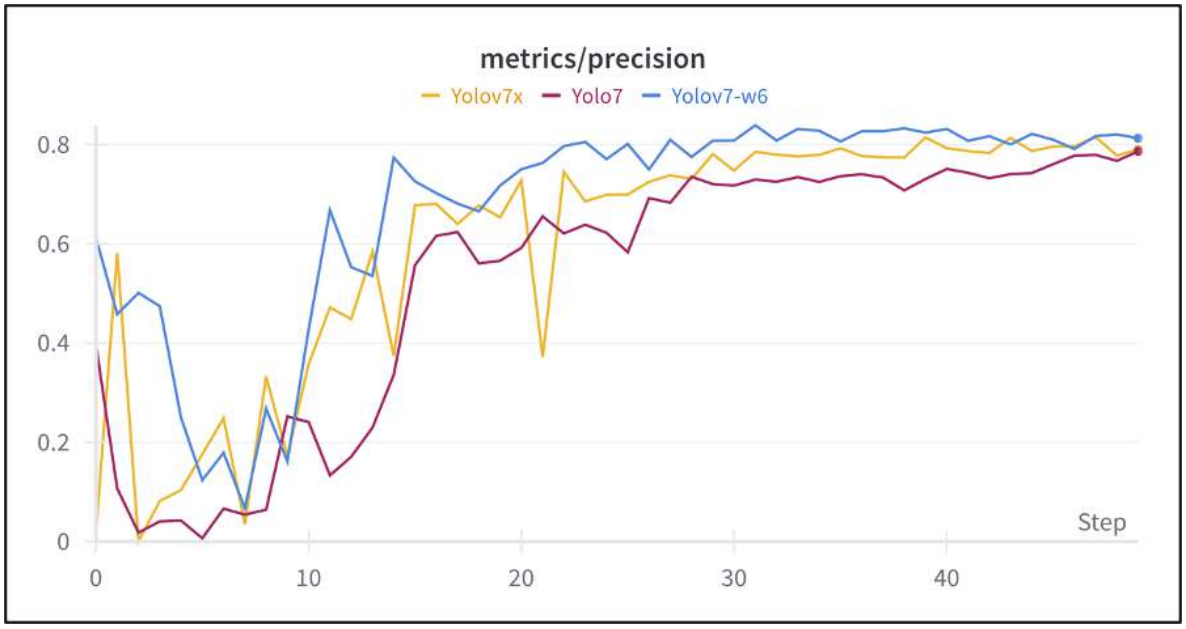

Рисунок 17 – График метрики precision

На рисунке 18 отображено динамическое изменения метрики recall на протяжении 50 эпох, наилучший результат показала модель yolov7-w6.

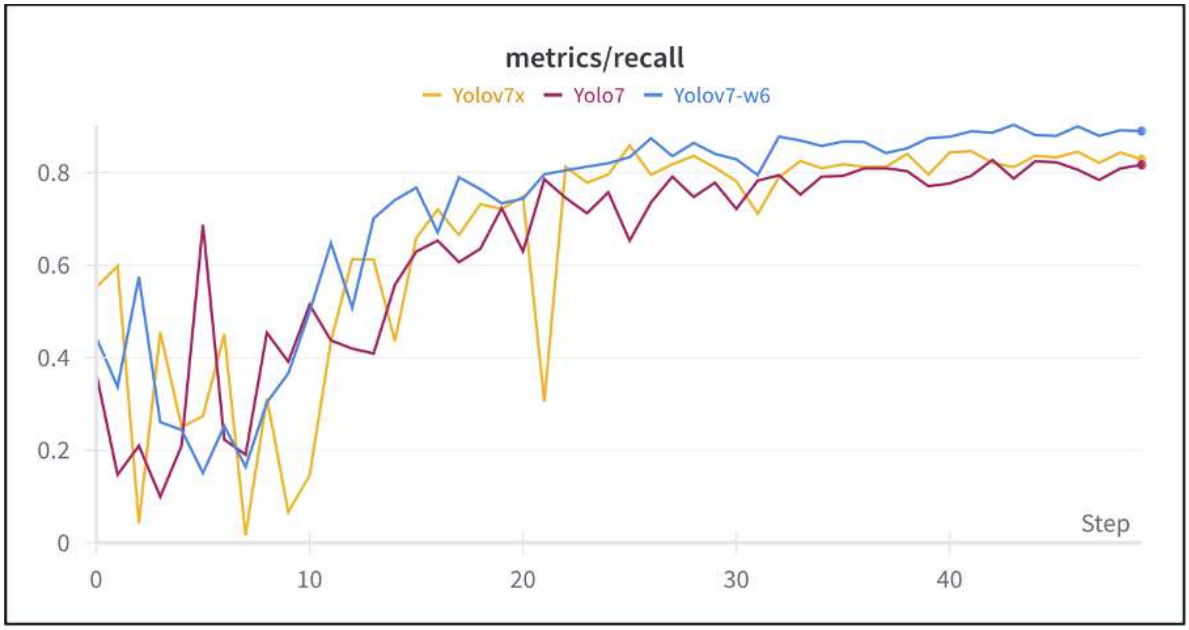

Рисунок 18 – График метрики recall

На рисунке 19 отображено динамическое изменения метрики recall на протяжении 50 эпох, наилучший результат показала модель yolov7-w6.

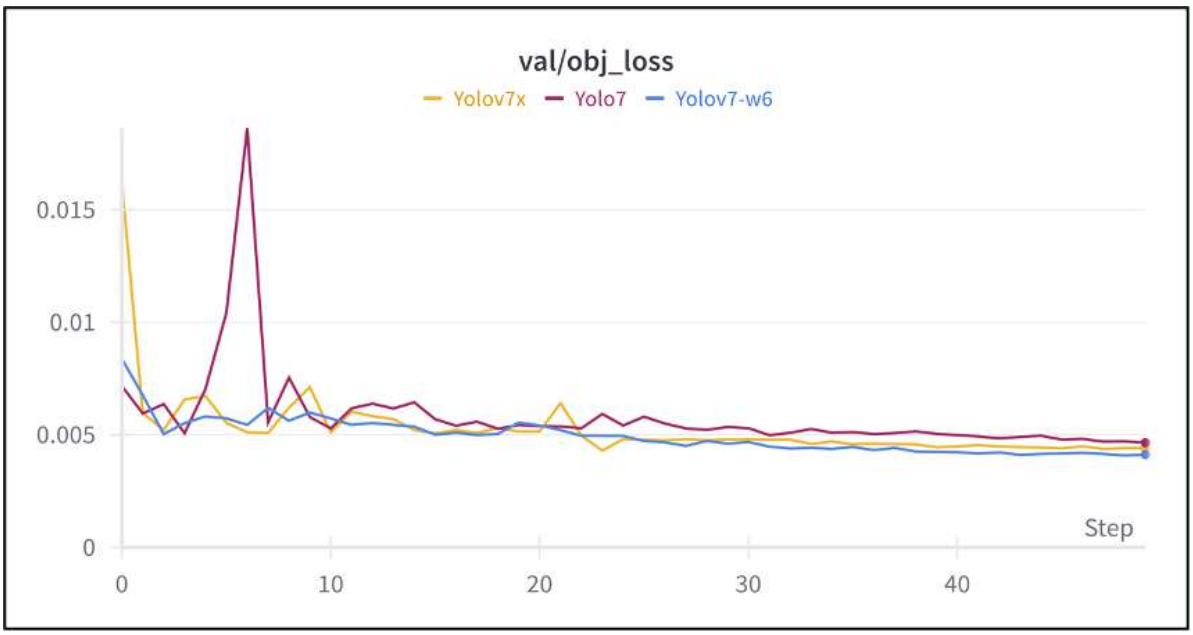

Рисунок 19 – График функции потерь на валидационной выборке

В результате обучения разных моделей, наилучший результат показала модель yolov7-w6, а именно метрика mAP 84,03%, метрика recall 88,89% и метрика precision 81,26%.

Для запуска детектирования и классификации была использована функция detect, которая поставляется вместе с YOLOv7. Результатом работы данной функции является обработанное изображение с размеченными объектами. Код для детектирования представлен в листинге 1 приложения А.

Поскольку в процессе детектирования нейронная сеть не запоминает информацию об объектах с изображения или видео, для это задачи был использована функция `track_video` из Deep SORT. Данная функция проставляет id каждому распознанному объекту, чтобы в дальнейшем понимать, появлялся ли данный объект раннее в кадре, таким образом происходит отслеживание объектов. Результатом работы данной функции является обработанное видео с результатом трекинга объектов на нем. Код для трекинга представлен в листинге 2 приложения А.

4.3. Реализация серверной части веб-приложения

Серверная часть веб-приложения была написана на языке Python и фреймворке FastApi. Код серверной части веб-приложения представлен в приложении Б. Серверная часть отвечает за прием файлов изображений или файла видео и передачи его в нейросетевую модель для детектирования или трекинга водителей электросамокатов, а после обработки она возвращает результат на клиентскую часть. Логика серверной части поделена на 3 основных файла: main.py, video.py, images.py и 2 вспомогательных файла files_validator.py и delete_output_files.py.

В файле main.py прописана логика запуска приложения и подключения роутингов video и images.

В файле files_validator.py находятся вспомогательные функции для валидации расширения загруженных файлов. Код валидации расширения файлов изображений и видео представлены в листингах 1 и 2.

Листинг 1 – Функция валидации расширения загруженных файлов изображений

```
def files_validator_detect(files: List[UploadFile]):
    for file in files:
        _, file_extension = os.path.splitext(file.filename)
        if file_extension != '.jpg' and file_extension != '.png' and
file_extension != '.jpeg':
            raise HTTPException(status_code=400,
detail="Расширения файлов должно быть PNG или JPG")
```

Листинг 2 – Функция валидации расширения загруженного файла видео

```
def file_validator_tracking(file: UploadFile):
    _, file_extension = os.path.splitext(file.filename)
    if file_extension != '.mp3' and file_extension != '.mp4':
        raise HTTPException(status_code=400, detail="Расширения файла
должно быть MP4")
```

Как видно из листингов 1 и 2, в случае неправильно расширения загруженного файла, серверная часть вернет ошибку со статусов 400 и соответствующий текст ошибки на клиентскую часть.

В файле video.py находится реализация эндпоинта для обработки видео (POST) /video/track, который принимает в качестве входных параметров

файл в формате form-data, затем этот файл передается в функцию для валидации (листинг 2), если файл имеет допустимое расширение, он передается в нейросетевую модель для трекинга, после чего, обработанное видео с результатами трекинга сохраняется в папку video/output и кодируется с помощью Base64, затем закодированное видео возвращается на клиентскую часть.

Также в этом файле находится реализация эндпоинта для скачивания обработанного видео (GET) /video/download-zip, который берет файл видео из папки video/output, формирует zip архив и возвращает на клиентскую часть.

В файле images.py находится реализация эндпоинта для обработки изображений (POST) /images/detect, который принимает в качестве входных параметров список файлов в формате form-data, затем этот список файлов передается в функцию для валидации (листинг 1), если все файлы имеют допустимое расширение, они по очереди передаются в нейросетевую модель для детектирования, после чего обработанное изображение сохраняется в папку images/output и кодируется с помощью Base64, после чего закодированное изображение добавляется в массив. После того как все изображения пройдут через вышеописанный процесс, массив закодированных изображений возвращается на клиентскую часть.

Также в этом файле находится реализация эндпоинта для скачивания обработанных изображений (GET) /images/download-zip, который берет файлы изображений из папки images/output, формирует zip архив и возвращает на клиентскую часть.

В файле delete_output_files.py находится вспомогательная функция для очистки дискового пространства, которая очищает папки images/output и video/output. Данная функция вызывается при каждом запуске приложения. Код для очистки дискового пространства представлен в листинге 3.

Листинг 3 – Функция для очистки дискового пространства

```
def delete_files_in_output():
  output_folder_path_list = ['images/output', 'video/output']
   for folder in output_folder_path_list:
    for filename in os.listdir(folder):
     file_path = os.path.join(folder, filename)
     try:
       if os.path.isfile(file_path):
        os.remove(file_path)
     except Exception as e:
      print (f'Ошибка при удалении файла {file_path}. {e}')
```

4.4. REST API

В серверной части приложения имеются API методы для взаимодействия с клиентской частью. Ниже представлены эти методы с их кратким описанием.

(POST) /images/detect

Данный метод принимает на вход список файлов. Этот список файлов передается в модель YOLOv7 для обработки, после обработки формируется массив байтов обработанных изображений. В случае успешной работы метод вернет код 200 и вернет массив байтов изображений. В случае не удачной обработки изображения, метод вернет HTTPException с кодом ошибки 400 и сообщение «Не удалось обработать изображение».

(GET) /images/download-zip

Данный метод предназначен для скачивания zip архива с обработанными изображениями. В случае успешной работы метод вернет код 200 и zip архив. В случае не удачной работы метода он вернет HTTPException с кодом ошибки 400 и сообщение «Не удалось скачать архив файл с результатами».

(POST) /video/track

Данный метод принимает на вход файл. Этот файл передается в метод трекинга Deep SORT для обработки, после обработки формируется строка байтов обработанного видео. В случае успешной работы метод вернет код 200 и вернет массив байтов изображений. В случае не удачной обработки

видео, метод вернет HTTPException с кодом ошибки 400 и сообщение «Не удалось обработать видео».

(GET) /video/download-zip

Данный метод предназначен для скачивания zip архива с обработанными изображениями. В случае успешной работы метод вернет код 200 и zip архив. В случае не удачной работы метода он вернет HTTPException с кодом ошибки 400 и сообщение «Не удалось скачать архив файл с результатами».

4.5. Реализация клиентской части веб-приложения

Клиентская часть веб-приложения была написана на языке Typescript и фреймворке Nest.js. Пользователю доступно две страницы, одна страница для детектирования, а другая для трекинга. Внешний вид страницы детектирования представлена на рисунке 20.

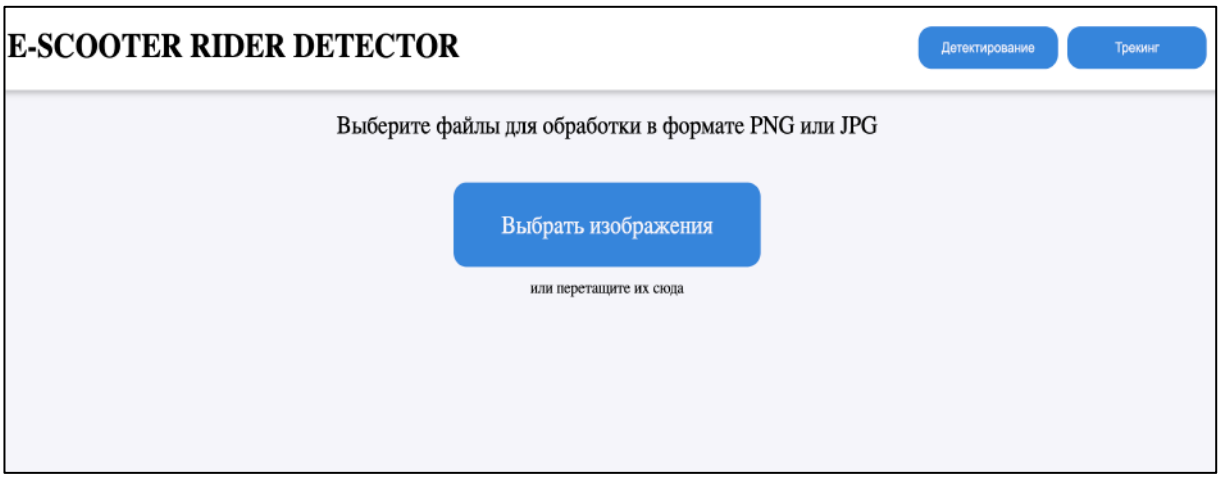

Рисунок 20 – Страница детектирования, интерфейс загрузки

На странице детектирования пользователя встречает интерфейс для загрузки файлов изображений. Код страницы детектирования представлен в листинге 5 приложения В. Пользователь может нажать на кнопку «Выбрать изображения» и выбрать одно или несколько изображений в файловом менеджере, или перетащить необходимые файлы в зону для загрузки. Если пользователь попробует загрузить файлы неверно формата, то он увидит

следующие окно с ошибкой «Расширение файлов должно быть PNG или JPG». Окно с текстом ошибки представлено на рисунке 21.

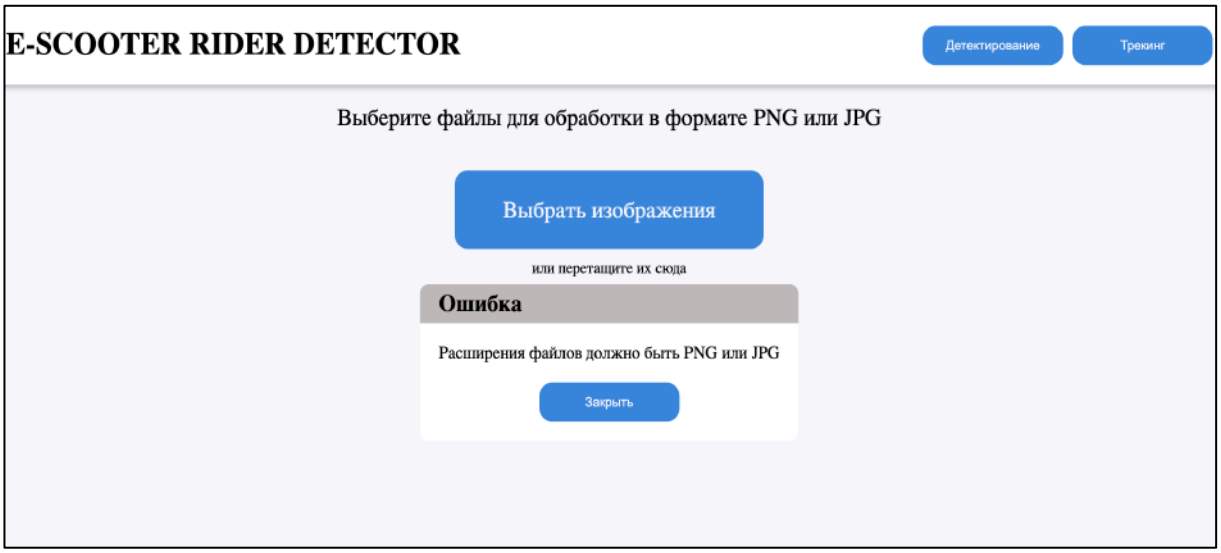

Рисунок 21 – Ошибка неверного формата загруженных файлов

Если же пользователь загрузил все файлы верного формата, то после успешной обработки он увидит интерфейс для просмотра обработанных изображений с результатами детектирования, который представлен на рисунке 22.

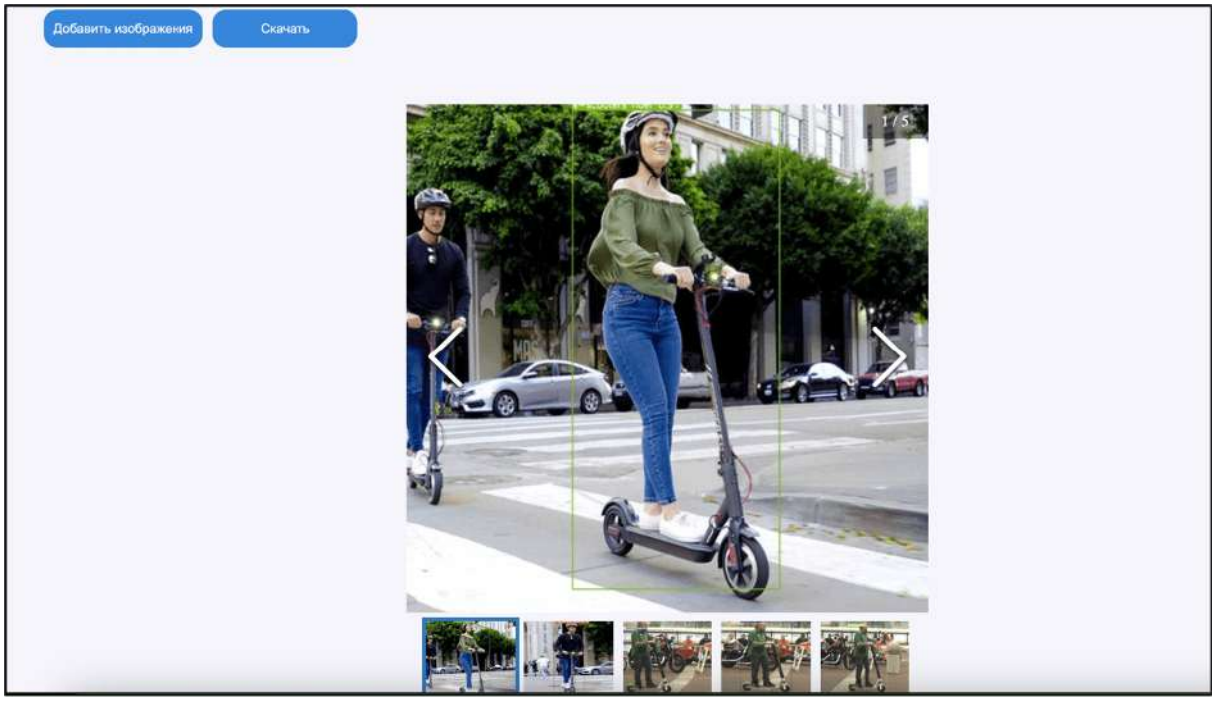

Рисунок 22 – Интерфейс просмотра обработанных изображений

В интерфейсе просмотра обработанных изображений пользователь может посмотреть все обработанные изображений с результатами детектирования. Также у пользователя появляется меню действий с помощью которого он может скачать zip архив с результатами или добавить еще изображений для обработки. После того как в веб-приложение было загружено хоть одно изображения, у пользователя появляется возможно переключения между интерфейсом загрузки и просмотра.

Внешний вид страницы трекинга представлена на рисунке 23. Код страницы трекинга представлен в листинге 6 приложения В. На странице трекинга пользователя встречает интерфейс для загрузки файла видео. Пользователь может нажать на кнопку «Выбрать видео» и выбрать одно видео в файловом менеджере, или перетащить файл в зону для загрузки.

Рисунок 23 – Страница трекинга, интерфейс загрузки

Если пользователь попробует загрузить файлы неверно формата, то он увидит следующие окно с ошибкой «Расширение видео должно быть MP4». Окно с текстом ошибки представлено на рисунке 24.

Рисунок 24 – Ошибка неверного формата загруженного файла видео

Если же пользователь загрузил файл видео верного формата, то после успешной обработки он увидит интерфейс для просмотра обработанного видео с результатами трекинга, который представлен на рисунке 25.

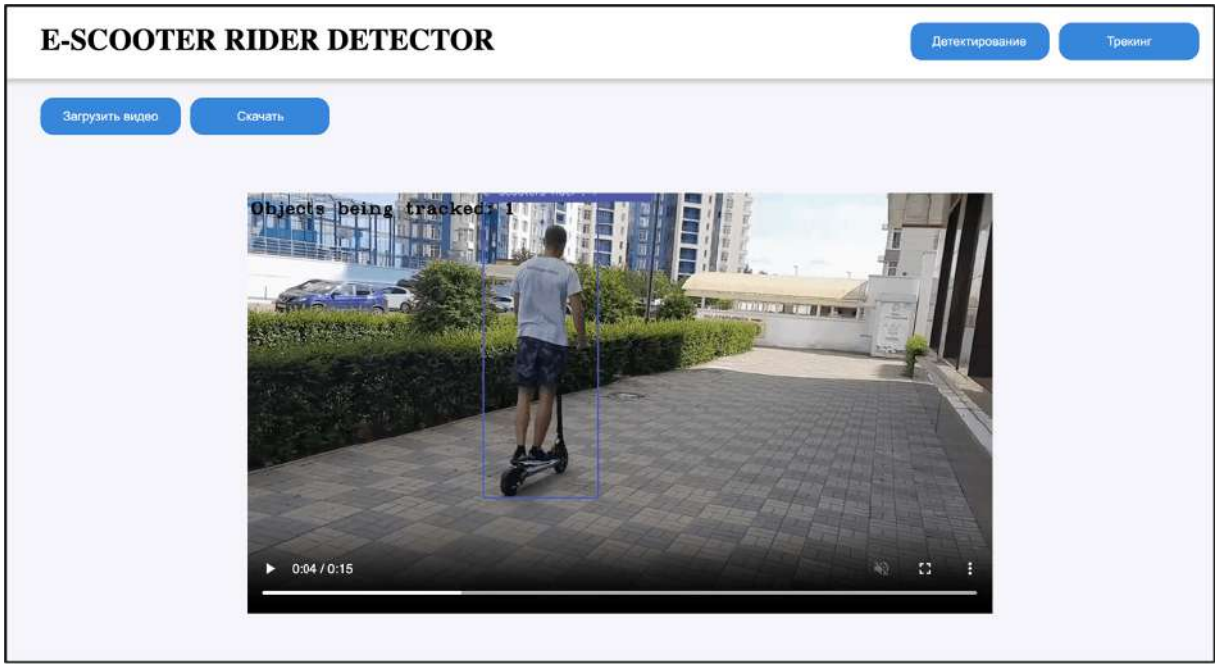

Рисунок 25 – Интерфейс просмотра обработанного видео

В интерфейсе просмотра обработанного видео пользователь может посмотреть результат трекинга на видео. Также у пользователя появляется

меню действий с помощью которого он может скачать zip архив с результатом или загрузить новое видео для обработки.

Также во время обработки изображений или видео, пользователю показывается значок обработки с подписью «Идет обработка...». Вид веб-приложения во время обработки представлен на рисунке 26.

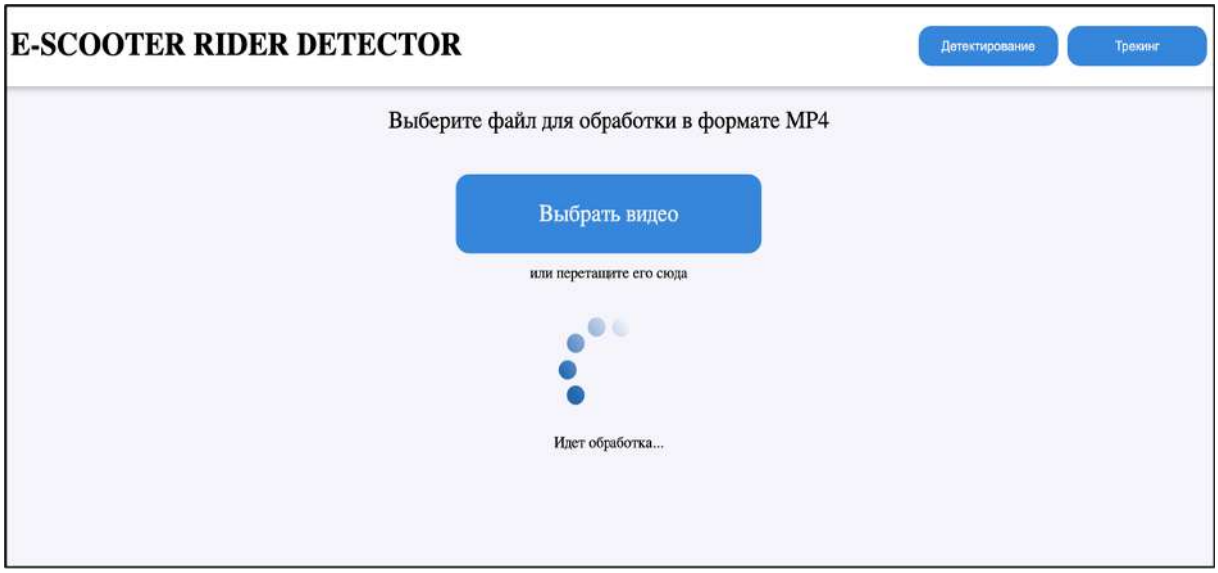

Рисунок 26 – Процесс обработки

Вывод по четвертой главе

В данной главе была реализована нейросетевая модель для детектирования и трекинга электросамокатов в транспортном потоке. Она была обучена на собранном наборе данных. Также было реализовано веб-приложение для демонстрации работы нейросетевой модели. Веб-приложение полностью соответствует функциональным и нефункциональным требованиям.

5. ТЕСТИРОВАНИЕ

5.1. Тестирование нейронной сети

Для тестирования работы нейронной сети были использованы разные модели, поставляемые в YOLOv7, а именно YOLOv7, YOLOv7x и YOLOv7-w6. Данные модели были обучены на разном количестве эпох, а для сравнения качества работы разных моделей была взята метрика mAP (mean Average Precision), которая измеряет точность распознавания объектов по всем классам, а именно вычисляет среднюю от интеграла функции зависимости precision от recall. Метрика recall – показывает, насколько точны предсказания модели. Метрика precision – показывает долю предсказаний, которые действительно оказались правдивыми. Сравнение качества работы разных моделей YOLOv7 представлено в таблицах 2–4.

Таблица 2 – Сравнение качества работы разных моделей по метрике mAP

Модель / **Количество эпох**	**YOLOv7**	**YOLOv7x**	**YOLOv7-w6**
15	54,52%	66,06%	75,32%
30	73,6%	76,02%	81,79%
50	79,72%	81,17%	84,03%

Таблица 3 – Сравнение качества работы разных моделей по метрики recall

Модель / **Количество эпох**	**YOLOv7**	**YOLOv7x**	**YOLOv7-w6**
15	62,95%	65,92%	76,78%
30	72,15%	78,13%	82,88%
50	81,17%	82,88%	88,89%

Таблица 4 – Сравнение качества работы разных моделей по метрике precision

Модель / **Количество эпох**	**YOLOv7**	**YOLOv7x**	**YOLOv7-w6**
15	55,69%	67,77%	72,57%
30	71,74%	74,76%	80,81%
50	78,64%	79,01%	81,26%

Результаты экспериментов показали, что наиболее качественный результат показывает модель YOLOv7-w6 при обучении на 50 эпох. Данная модель показала точность метрики mAP 84,03%, метрики recall 88,89% и метрики precision 81,26%.

5.2. Функциональное тестирование веб-приложения

Для проверки работоспособности веб-приложения было проведено функциональное тестирование. Результаты функционального тестирования представлены в таблице 5.

Таблица 5 – Результаты функционального тестирования приложения

№	Название теста	Входные данные	Ожидаемый результат	Результат
1	Загрузка изображения для детектирования с помощью зоны для загрузки или кнопки «Выбрать изображения»	Пользователь заходит на страницу детектирования, перетаскивает изображение в зону для загрузки или нажимает на кнопку «Выбрать изображения» и выбирает файлы изображений	Во время обработки изображения, на странице появится значок обработки, после завершения обработки, пользователю показывается интерфейс для просмотра обработанного изображения	Пройден
2	Отображение ошибки на странице детектирования при загрузке файла с неправильным форматом	Пользователь заходит на страницу детектирования, перетаскивает файл с неправильным расширение в зону для загрузки или нажимает на кнопку «Выбрать изображения» и выбирает файл с неправильным расширением	Пользователю показывается всплывающие окно с текстом ошибки «Расширение файлов должно быть PNG или JPG».	Пройден
3	Скачивание zip архива с результатами обработанных изображений на странице детектирования	Пользователь находится на странице детектирования в интерфейсе для просмотра результатов и нажимает на кнопку «Скачать»	Скачивается zip файла, в котором содержатся все обработанные изображения	Пройден

Окончание таблицы 5

№	Название теста	Входные данные	Ожидаемый результат	Результат
4	Дозагрузка изображений на странице детектирования	Пользователь находится на странице детектирования, он до этого загрузил одно изображение и находится в интерфейсе для просмотра результата, пользователь нажимает на кнопку добавить изображений, он переходит в интерфейс для загрузки, и загружает еще одно изображение	Во время обработки изображения, на странице появится значок обработки, после завершения обработки, пользователю показывается интерфейс для просмотра обработанных изображений, где к предыдущему результату добавляется новое обработанного изображения	Пройден
5	Загрузка видео для трекинга с помощью зоны для загрузки или кнопку «Выбрать видео»	Пользователь заходит на страницу трекинга, перетаскивает видео в зону для загрузки или нажимает на кнопку «Выбрать видео» и выбирает файл видео	Во время обработки видео, на странице появится значок обработки, после завершения обработки, пользователю показывается интерфейс для просмотра обработанного видео	Пройден
6	Отображение ошибки на странице трекинга при загрузке файла с неправильным форматом	Пользователь заходит на страницу трекинга и загружает файл с неправильным расширение	Пользователю показывается всплывающие окно с текстом ошибки «Расширение файлов должно быть MP4»	Пройден
7	Скачивание zip архива с результатами обработанного видео на странице трекинга	Пользователь находится на странице трекинга в интерфейсе для просмотра результатов и нажимает на кнопку «Скачать»	Скачивается zip файла, в котором содержится обработанное видео	Пройден

5.3. Автоматизированное тестирование веб-приложения

Автоматизированное тестирование веб-приложения проводилось с помощью Cypress [22]. Cypress это инструмент для тестирования пользовательского интерфейс. Веб-приложение было протестировано с помощью нескольких видов тестов, а именно компонентными тестами и E2E (end-to-end)

тестами. Компонентное тестирование – это вид тестирования, который позволяет протестировать функционал отдельно взятых частей приложения в изоляции от всего приложения. Обычно для такого вида тестов используются тестовые данные. E2E тестирование это вид тестирования в котором приложение тестируется полностью, а не какая-то его отдельная часть. Обычно для такого вида тестов используются реальные данные.

При запуске cypress открывается окно с выбором вида тестирования, оно представлено на рисунке 27.

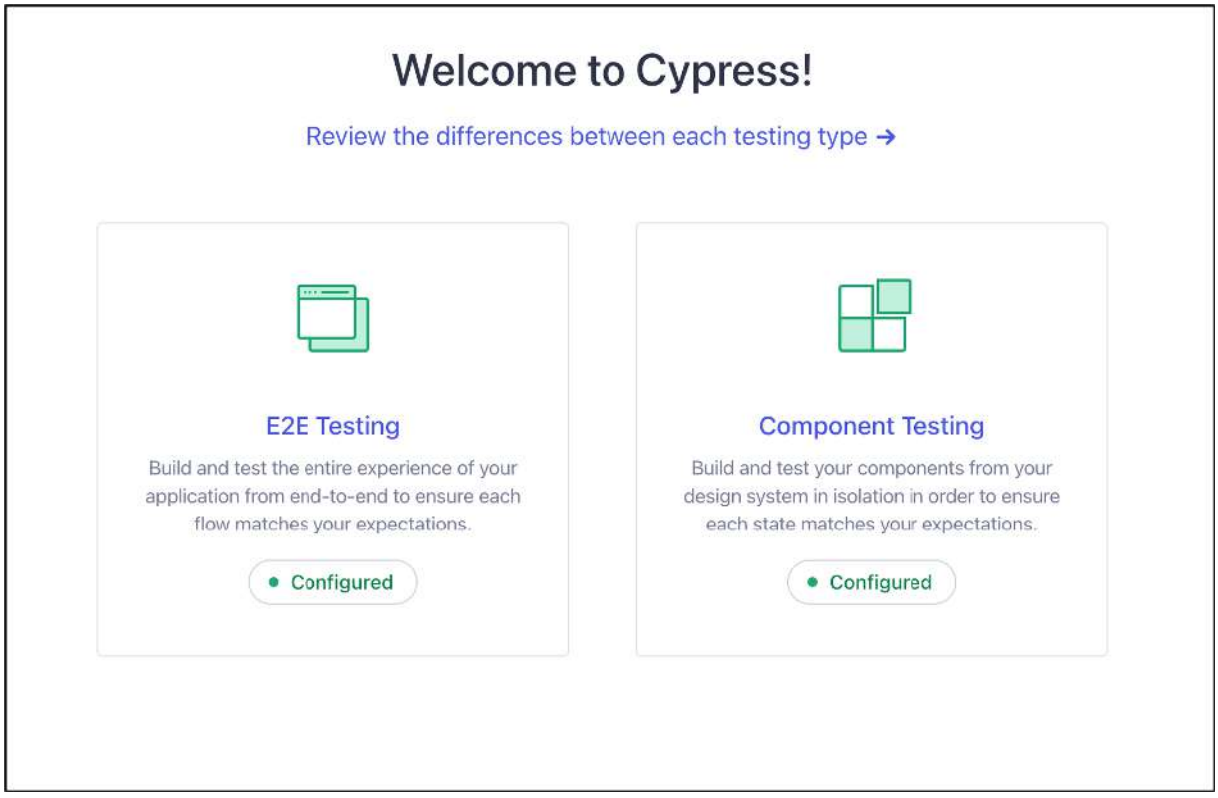

Рисунок 27 – Окно с выбор вида тестирования

После выбора вида тестирования, необходимо выбрать браузер, в котором будет тестироваться приложение. Окно с выбором браузера представлено на рисунке 28.

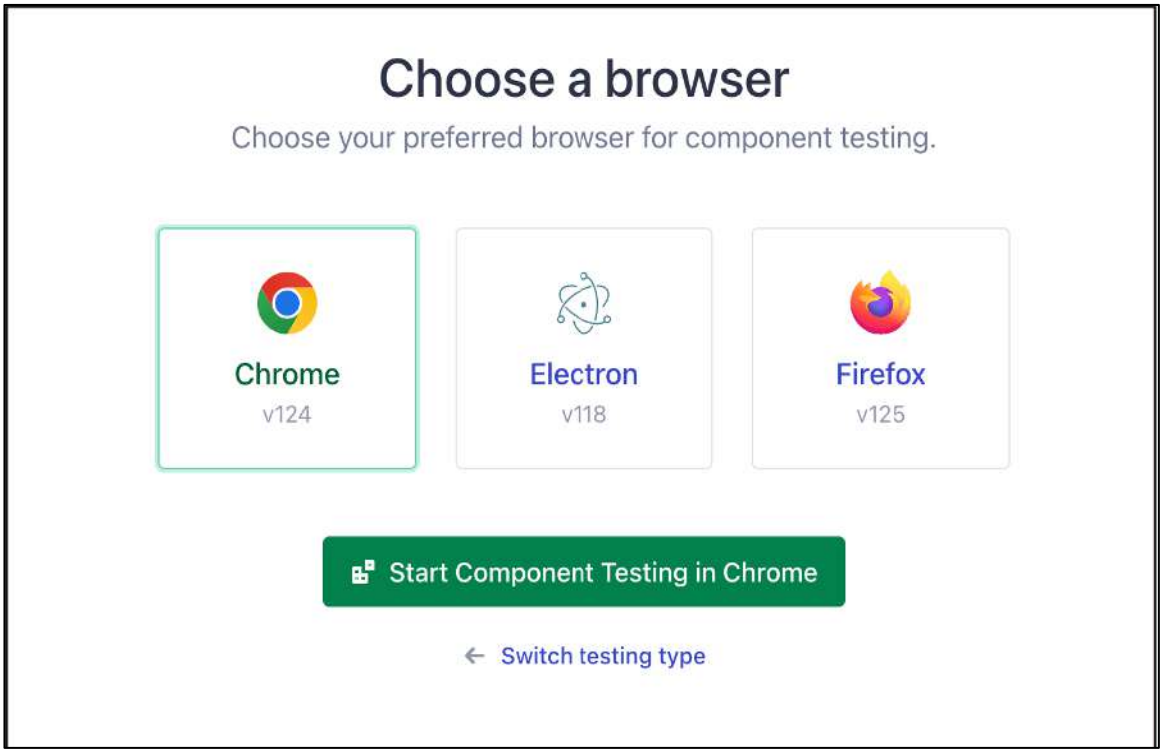

Рисунок 28 – Окно с выбором браузера

После выбора браузера, откроется веб-интерфейс, в котором можно выбрать тест и запустить его, интерфейс с выбором тестов представлен на рисунке 29.

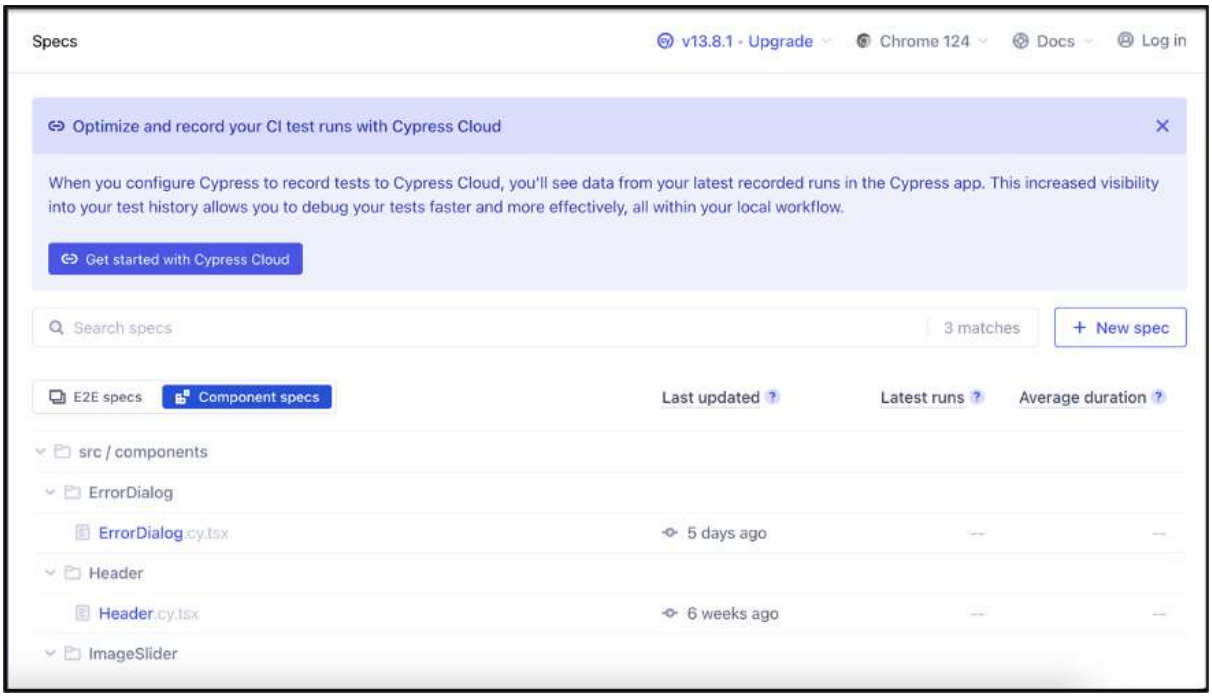

Рисунок 29 – Интерфейс выбора тестов

Для демонстрации интерфейса выполнения теста был выбран тест для `ErrorDialog` компоненты. Как видно на рисунке 30, в правой части интерфейса окошко, в котором отрисовывается тестируемая компонента, а в левой части находится список тестов с результатом выполнения, все тесты успешно выполнены.

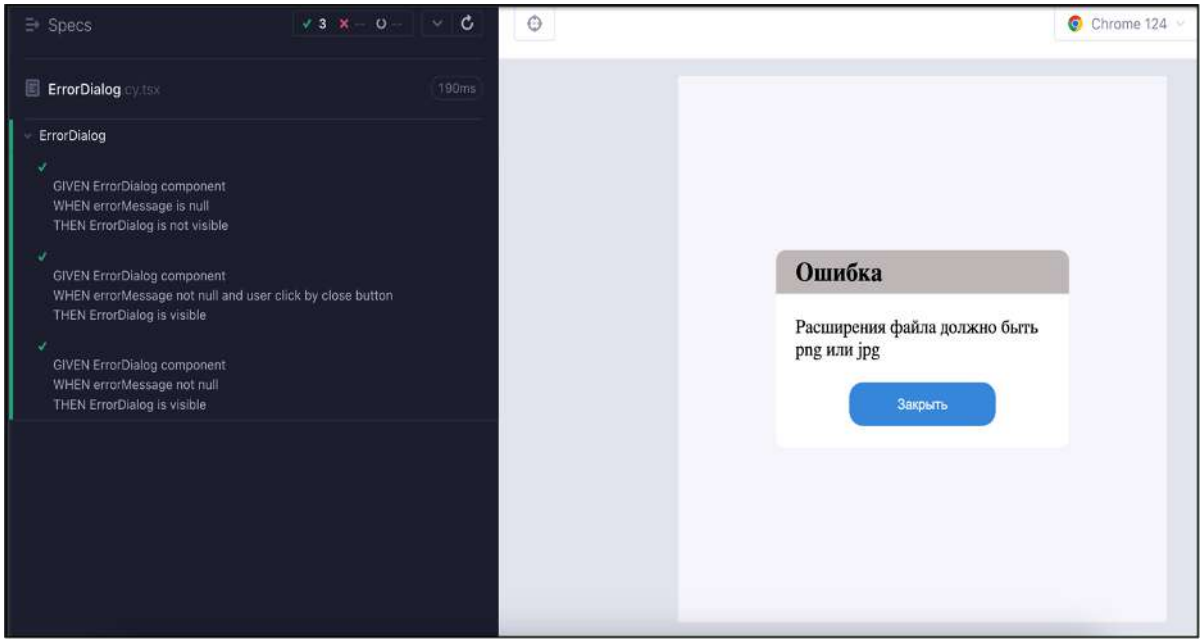

Рисунок 30 – Тест на компонент ErrorDialog

Код компонентного теста для `ErrorDialog` представлен в листинге 4.

Листинг 4 – Код компонентного теста для `ErrorDialog`

```
describe('ErrorDialog', () => {
  it(`
  GIVEN ErrorDialog component
  WHEN errorMessage not null and user click by close button
  THEN ErrorDialog is visible
  `, () => {
    mountComponent({
      errorMessage: 'Расширения файла должно быть png или jpg',
    });

    cy.get('.error-dialog')
      .should('exist');

    cy.get('.error-dialog__button').click();
    cy.get('.error-dialog')
      .should('not.exist');
  });


  it(`
  GIVEN ErrorDialog component
  WHEN errorMessage not null
  THEN ErrorDialog is visible
```

```
  `, () => {
    mountComponent({
      errorMessage: 'Расширения файла должно быть png или jpg',
    });

    cy.get('.error-dialog')
      .should('exist');

    cy.contains('Расширения файла должно быть png или jpg');
  });
});

function mountComponent({
  errorMessage,
}: {
  errorMessage: string | null;
}) {
  cy.mount(
    <ErrorDialog message={errorMessage} />,
  );
}
```

Код E2E теста для страницы детектирования представлен в листинге 5. Данный тест имитирует действия пользователя, который заходит на страницу детектирования и загружает одно изображение, получает результат обработанного изображение и после этого загружает еще одно изображение, и затем скачивает zip архив с результатами детектирования.

Листинг 5 – Код E2E теста для страницы детектирования

```
describe('Detect Page', () => {
  it(`
  GIVEN detect page
  WHEN the user visits the detect page upload one image, then another image
and then clicks on the download button
  THEN downloads a zip file with the results
  `, () => {
    cy.viewport(800, 800);
    cy.visit('http://localhost:3000');
    cy.intercept('POST', '/images/detect').as('detectImage');

    cy.get('.dropzone').selectFile([
      'cypress/fixtures/test.jpg',
    ], {
      action: 'drag-drop',
    });

    cy.get('.preloader')
      .should('exist');
    cy.get('.image-slider')
      .should('not.exist');

    cy.wait('@detectImage', { requestTimeout: 40000 });

    cy.get('.preloader')
      .should('not.exist');
```

```
    cy.get('.image-slider')
      .should('exist');

    cy.get('.action-bar__add-button').click();
    cy.get('.dropzone').selectFile([
      'cypress/fixtures/test.jpg',
    ], {
      action: 'drag-drop',
    });
    cy.wait('@detectImage', { requestTimeout: 40000 });
    cy.get('.image-slider')
      .should('exist');
    cy.contains('Скачать').click();
    cy.readFile('cypress/downloads/results.zip');
  });
});
```

Вывод по пятой главе

В данной главе было проведено тестирование нейронной сети, тестирование проводилось на разных моделях YOLOv7, наилучший результат показала модель YOLOv7-w6. Также было проведено функциональное тестирование, все тесты успешно проходят. Помимо этого, было проведено автоматизированное тестирование веб-приложения с помощью инструмента cypress, веб-приложение было протестировано с помощью двух видов тестирования, компонентного и E2E, все тесты успешно проходят.

ЗАКЛЮЧЕНИЕ

В соответствии с целью данной работы была разработана нейросетевая модель для распознавания и отслеживания электросамокатов в транспортном потоке. Помимо этого, были выполнены следующие задачи:

1) произведен обзор научной литературы и существующих приложений по предметной области;
2) собран и размечен набор данных;
3) спроектировано веб-приложение;
4) обучена и протестирована нейросетевая модель для детектирования и отслеживания электросамокатов;
5) разработано веб-приложение;
6) протестирована работа веб-приложения;
7) проведен анализ полученных результатов.

Для улучшения работы нейросетевой модели можно сделать следующие действия: увеличить набор данных, добавить больше разнообразия данных, дообучить нейросетевую модель, изменить топологи модели, попробовать подобрать параметры, с которыми модель выдаст более лучшие результаты.

ЛИТЕРАТУРА

1. Вьюгин В.В. Математические основы теории машинного обучения и прогнозирования. // М.: 2013. – 387 с.

2. Барский А.Б. Нейронные сети: распознавание, управление, принятие решений. // М.: Финансы и статистика, 2004. – 176 с.

3. Воронин В.В. Теория и практика машинного обучения: учебное пособие. / В.В. Воронина, А.В. Михеев, Н.Г. Ярушкина, К.В. Святов // Ульяновск: УлГТУ, 2017. – 290 с.

4. Li C., Wang Y., Liu X. An improved YOLOv7 lightweight detection algorithm for obscured pedestrians. // Sensors (Basel), 2023. – Т. 23, № 13. – С. 5912–5926.

5. Zhang M. Cyclist detection and tracking based on multi-layer laser scanner. // Hum. – centric Comput. Inf. Sci, 2020. – Т. 10, № 1. С. 1–18.

6. Tamura M., Yoshinaga T. Segmentation-based bounding box generation for omnidirectional pedestrian detection. // Vis. Comput, 2023 – С. 21–28.

7. Gilroy S. E-scooter rider detection and classification in dense urban environments. // 2022 – С. 102–111.

8. Apurv K., Tian R., Sherony R. Detection of E-scooter riders in naturalistic scenes. // 2021 – С. 50–57.

9. Gawande U., Hajari K., Golhar Y. Real-time deep learning approach for pedestrian detection and suspicious activity recognition. // Procedia Computer Science, 2023. – Т. 218 – С. 2438–2447.

10. Tang F., Yang F., Tian X. Long-distance person detection based on Yolov7. // Electronics, 2023. – Т. 12, № 6. – С. 1502–1516.

11. Официальный сайт Pytorch. [Электронный ресурс] URL: https://pytorch.org/ (дата обращения: 03.03.2024 г.).

12. Wang C.-Y., Bochkovskiy A., Liao H.-Y. M. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. // 2022 – 15 с.

13. Chang B. R., Tsai H.-F., Hsieh C.-W. Accelerating the response of self-driving control by using rapid object detection and steering angle prediction. // Electronics (Basel), 2023. – Т. 12. № 10. – С. 2161–2198.

14. Liu S., Lu Qi., Haifang Qin. Path Aggregation Network for instance segmentation. // 2018 –11 с.

15. Набор данных E-Scooter Rider. [Электронный ресурс] URL: http://situated-intent.net/e-scooter_dataset/ (дата обращения: 10.03.2024 г.).

16. Набор данных Occluded E-Scooter Rider Detection Dataset. [Электронный ресурс] URL: https://github.com/SGPHD/Occluded-E-Scooter-Rider-Dataset?tab=readme-ov-file (дата обращения: 10.03.2024 г.).

17. Официальный сайт Roboflow. [Электронный ресурс] URL: https://roboflow.com/ (дата обращения: 10.03.2024 г.).

18. Github репозиторий алгоритма Deep SORT. [Электронный ресурс] URL: https://github.com/nwojke/deep_sort?tab=readme-ov-file (дата обращения 15.03.2024 г.).

19. Wojke N., Bewley A., Paulus D. Simple online and Realtime Tracking with a deep association metric. // 2017 – 5 с.

20. Официальная документация Next.js. [Электронный ресурс] URL: https://nextjs.org/ (дата обращения: 22.03.2024 г.).

21. Официальная документация TypeScript. [Электронный ресурс] URL: https://www.typescriptlang.org/ (дата обращения: 22.03.2024 г.).

22. Официальная документация Cypress. [Электронный ресурс] URL: https://www.cypress.io/ (дата обращения: 22.03.2024 г.).

23. Официальная документация Python. [Электронный ресурс] URL: https://www.python.org/ (дата обращения: 22.03.2024 г.).

24. Официальная документация FastApi. [Электронный ресурс] URL: https://fastapi.tiangolo.com/ (дата обращения 24.03.2024 г.).

25. Google Colaboratory. [Электронный ресурс] URL: https://colab.research.google.com/?hl=ru_RU (дата обращения 24.03.2024 г.).

ПРИЛОЖЕНИЯ

Приложение А. Код для детектирования и трекинга

Листинг 1 – Код для детектирования

```
def detect(self, source, plot_bb:bool =True):
    img, im0 = self.load_image(source)
    img = torch.from_numpy(img).to(self.device)
    img = img.half() if self.half else img.float()
    img /= 255.0
    if img.ndimension() == 3:
        img = img.unsqueeze(0)

    pred = self.model(img, augment=False)[0]
    pred = non_max_suppression(pred, self.conf_thres, self.iou_thres, clas-
ses=self.classes, agnostic=self.agnostic_nms)

    if self.classify:
        pred = apply_classifier(pred, self.modelc, img, im0

    det = pred[0]
    if len(det):
        det[:, :4] = scale_coords(img.shape[2:], det[:, :4],
im0.shape).round()
        for *xyxy, conf, cls in reversed(det):

            if plot_bb:
                label = f'{self.names[int(cls)]} {conf:.2f}'
                plot_one_box(xyxy, im0, label=label, color=self.col-
ors[int(cls)], line_thickness=1)


        return im0 if plot_bb else det.detach().cpu().numpy()

    return im0 if plot_bb else None
```

Листинг 2 – Код для трекинга

```
def track_video(self,video:str, output:str, skip_frames:int=0,
show_live:bool=False, count_objects:bool=False, verbose:int = 0):
    try:
        vid = cv2.VideoCapture(int(video))
    except:
        vid = cv2.VideoCapture(video)

    out = None
    if output:
        width = int(vid.get(cv2.CAP_PROP_FRAME_WIDTH))
        height = int(vid.get(cv2.CAP_PROP_FRAME_HEIGHT))
        fps = int(vid.get(cv2.CAP_PROP_FPS))
        codec = cv2.VideoWriter_fourcc(*"XVID")
        out = cv2.VideoWriter(output, codec, fps, (width, height))

    frame_num = 0
    while True:
        return_value, frame = vid.read()
        if not return_value:
            print('Video has ended or failed!')
            break
        frame_num +=1
```

```
        if verbose >= 1:start_time = time.time()
        yolo_dets = self.detector.detect(frame.copy(), plot_bb = False)
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        if yolo_dets is None:
            bboxes = []
            scores = []
            classes = []
            num_objects = 0

        else:
            bboxes = yolo_dets[:,:4]
            bboxes[:,2] = bboxes[:,2] - bboxes[:,0]
            bboxes[:,3] = bboxes[:,3] - bboxes[:,1]

            scores = yolo_dets[:,4]
            classes = yolo_dets[:,-1]
            num_objects = bboxes.shape[0]
        names = []
        for i in range(num_objects):
            class_indx = int(classes[i])
            class_name = self.class_names[class_indx]
            names.append(class_name)

        names = np.array(names)
        count = len(names)

        if count_objects:
            cv2.putText(frame, "Objects being tracked: {}".format(count),
(5, 35), cv2.FONT_HERSHEY_COMPLEX_SMALL, 1.5, (0, 0, 0), 2)
        features = self.encoder(frame, bboxes)
        detections = [Detection(bbox, score, class_name, feature) for bbox,
score, class_name, feature in zip(bboxes, scores, names, features)]
        cmap = plt.get_cmap('tab20b')
        colors = [cmap(i)[:3] for i in np.linspace(0, 1, 20)]
        boxs = np.array([d.tlwh for d in detections])
        scores = np.array([d.confidence for d in detections])
        classes = np.array([d.class_name for d in detections])
        indices = preprocessing.non_max_suppression(boxs, classes,
self.nms_max_overlap, scores)
        detections = [detections[i] for i in indices]

        self.tracker.predict()
        self.tracker.update(detections)

        for track in self.tracker.tracks:
            if not track.is_confirmed() or track.time_since_update > 1:
                continue
            bbox = track.to_tlbr()
            class_name = track.get_class()

            color = colors[int(track.track_id) % len(colors)]
            color = [i * 255 for i in color]
            cv2.rectangle(frame, (int(bbox[0]), int(bbox[1])),
(int(bbox[2]), int(bbox[3])), color, 2)
            cv2.rectangle(frame, (int(bbox[0]), int(bbox[1]-30)),
(int(bbox[0])+(len(class_name)+len(str(track.track_id)))*17, int(bbox[1])),
color, -1)
            cv2.putText(frame, class_name + " : " +
str(track.track_id),(int(bbox[0]), int(bbox[1]-11)),0, 0.6,
(255,255,255),1, lineType=cv2.LINE_AA)
```

```
            if verbose == 2:
                print("Tracker ID: {}, Class: {},  BBox Coords (xmin, ymin,
xmax, ymax): {}".format(str(track.track_id), class_name, (int(bbox[0]),
int(bbox[1]), int(bbox[2]), int(bbox[3]))))
        if verbose >= 1:
            fps = 1.0 / (time.time() - start_time)
            if not count_objects: print(f"Processed frame no: {frame_num}
|| Current FPS: {round(fps,2)}")
            else: print(f"Processed frame no: {frame_num} || Current FPS:
{round(fps,2)} || Objects tracked: {count}")

        result = np.asarray(frame)
        result = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
        if output: out.write(result)

        if show_live:
            cv2.imshow("Output Video", result)
            if cv2.waitKey(1) & 0xFF == ord('q'): break

    cv2.destroyAllWindows()
```

Приложение Б. Код серверной части веб-приложения

Листинг 3 – Код метода `detect`

```
@router.post("/images/detect")
def detect(files: List[UploadFile]):
    files_validator_detect(files)
    try:
        detect_results = []

        for file in files:
            with open(f'images/input/{file.filename}', "wb") as buffer:
                shutil.copyfileobj(file.file, buffer)

                detector = Detector()
                detector.load_model('./weights/best.pt')

                input_path_to_files = f'images/input/{file.filename}'
                output_path_to_files = f'images/output/{file.filename}'

                detect_result = detector.detect(source=input_path_to_files)

            cv2.imwrite(output_path_to_files, detect_result)

            with open(output_path_to_files, "rb") as image_file:
                encoded_string = base64.b64encode(image_file.read())

            os.remove(input_path_to_files)

            detect_results.append(encoded_string)

        return detect_results
    except:
        raise HTTPException(status_code=400, detail='Не удалось обработать
изображение')
```

Листинг 4 – Код метода `track`

```
@router.post("/video/track")
def track(file: UploadFile):
    file_validator_tracking(file)
    try:
        with open(f'video/input/{file.filename}', "wb") as buffer:
            shutil.copyfileobj(file.file, buffer)

        detector = Detector()
        detector.load_model('./weights/best.pt')

        tracker =
YOLOv7_DeepSORT(reID_model_path="./deep_sort/model_weights/mars-
small128.pb", detector=detector)
        input_path_to_file = f'video/input/{file.filename}'
        output_path_to_file = f'video/output/{file.filename}'
        tracker.track_video(
            video=input_path_to_file,
            output=f'video/output/{file.filename}',
            show_live=False,
            skip_frames=0,
            count_objects=True,
            verbose=1
        )
```

Окончание листинга 4 приложения Б

```
        with open(output_path_to_file, "rb") as video_file:
            encoded_string = base64.b64encode(video_file.read())
        os.remove(output_path_to_file)
        os.remove(input_path_to_file)

        return encoded_string
    except:
        raise HTTPException(status_code=400, detail='Не удалось обработать
видео')
```

Приложение В. Код клиентской части веб-приложения

Листинг 5 – Код страницы детектирования

```
export default function DetectPage() {
  const [dataImage] = useState<string[]>([]);
  const [openDropZone, setOpenDropZone] = useState(true);
  const [isProcessDetection, setIsProcessDetection] = useState(false);
  const { setErrorMessage } = useErrorDialogContext();

  return (
    <>
      <Head>
        <title>E-SCOOTER RIDER DETECTOR</title>
        <meta name="viewport" content="width=device-width, initial-scale=1"
/>
        <link rel="icon" href="/favicon.ico" />
      </Head>
      <Layout>
        {dataImage.length !== 0 && (
          <ActionBar
            disabled={isProcessDetection}
            openDropZone={openDropZone}
            handleChangeStateDropZone={handleChangeStateDropZone}
          />
        )}
        {openDropZone && (
          <Dropzone
            onDrop={detect}
            disabled={isProcessDetection}
            dropText="или перетащите их сюда"
            description="Выберите файлы для обработки в формате PNG или
JPG"
            uploadButton={(
              <Uploader
                onChange={detect}
                disabled={isProcessDetection}
                text="Выбрать изображения"
                accept=".png, .jpg"
              />
            )}
          />
        )}
        { !openDropZone && dataImage.length !== 0 && <ImageSlider da-
taImageSrcList={dataImage} />}
      </Layout>
      {isProcessDetection && <Preloader />}
    </>
  );

  async function detect(files:File[]) {
    const formData = new FormData();
    for (const file of files) {
      formData.append('files', file);
    }

    setIsProcessDetection(true);

    try {
      const { data } = await axios.post<string[]>('http://lo-
calhost:8000/images/detect', formData);
```

```
      data.map((bytes) => dataImage.push(`data:im-
age/jpg;base64,${bytes}`));

      handleChangeStateDropZone();
    } catch (error) {
      if (axios.isAxiosError(error)) {
        setErrorMessage(error.response?.data.detail);
      }
    } finally {
      setIsProcessDetection(false);
    }
  }

  function handleChangeStateDropZone() {
    setOpenDropZone(!openDropZone);
  }
}
```

Листинг 6 – Код страницы трекинга

```
export default function TrackingPage() {
  const [videoSrc, setVideoSrc] = useState<string>();
  const [openDropZone, setOpenDropZone] = useState(true);
  const [isProcessTracking, setIsProcessTracking] = useState(false);
  const { setErrorMessage } = useErrorDialogContext();

  return (
    <>
      <Head>
        <title>E-SCOOTER RIDER DETECTOR</title>
        <meta name="viewport" content="width=device-width, initial-scale=1"
/>
        <link rel="icon" href="/favicon.ico" />
      </Head>
      <Layout>
        {videoSrc && (
          <ActionBar
            isDetect={false}
            disabled={isProcessTracking}
            openDropZone={openDropZone}
            handleChangeStateDropZone={handleChangeStateDropZone}
          />
        )}
        {openDropZone && (
          <Dropzone
            onDrop={track}
            multiple={false}
            dropText="или перетащите его сюда"
            disabled={isProcessTracking}
            description="Выберите файл для обработки в формате MP4"
            uploadButton={(
              <Uploader
                onChange={track}
                text="Выбрать видео"
                disabled={isProcessTracking}

                multiple={false}
                accept=".mp4"
              />
```

```
            )}
          />
        )}
        { !openDropZone && videoSrc && (
          <div>
            <Video videoSrc={videoSrc} />
          </div>
        )}
      </Layout>
      {isProcessTracking && <Preloader />}
    </>
  );

  async function track(file: File[]) {
    const formData = new FormData();

    formData.append('file', file[0]);

    setIsProcessTracking(true);

    try {
      const { data } = await axios.post<string>('http://lo-
calhost:8000/video/track', formData);
      setVideoSrc(`data:video/mp4;base64,${data}`);

      handleChangeStateDropZone();
    } catch (error) {
      if (axios.isAxiosError(error)) {
        setErrorMessage(error.response?.data.detail);
      }
    } finally {
      setIsProcessTracking(false);
    }
  }

  function handleChangeStateDropZone() {
    setOpenDropZone(!openDropZone);
  }
}
```