МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего образования

«ТЮМЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ НАУК

Кафедра программного обеспечения

РЕКОМЕНДОВАНО К ЗАЩИТЕ В ГЭК
Заведующий кафедрой
К.т.н., доцент
_____ М. С. Воробьева
25 июня 2022 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

магистерская диссертация

РАЗРАБОТКА СИСТЕМЫ МОНИТОРИНГА СОСТОЯНИЯ ПОКРЫТИЯ ПРОЕЗЖЕЙ ЧАСТИ

02.04.03 Математическое обеспечение и администрирование информационных систем

Магистерская программа «Разработка технологий Интернета вещей и больших данных»

Выполнил работу
студент 2 курса
очной формы обучения
Захаров Дмитрий Алексеевич

Научный руководитель
д.п.н., профессор
Захарова Ирина Гелиевна

Рецензент
к.ф.-м.н, доцент
Ступников Андрей Анатольевич

Тюмень
2022 год

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Суммарная протяженность автомобильных дорог в России на сегодняшний день составляет 1,553 млн. километров, 1,096 млн. км. из которых являются дорогами с твердым покрытием [1]. Такая большая протяженность дорог требует от государственных служб их поддержания в надлежащем состоянии.

Дороги особенно подвержены образованию выбоин во время перехода времен года, когда большое количество проточной воды разрушает базовые слои под покрытием, образуя слабые места. Когда машины проезжают по этим слабым местам, дорожное покрытие деформируется, появляются трещины и сколы, образуются дыры, которые становятся проблемой для любого автомобилиста.

Плохое состояние дорожного покрытия значительно влияет на условия движения транспортного средства: появляются вредные для водителя и автомобиля вибрации, существенно усложняются условия управления автомобилем, поскольку водителю требуется следить не только за движением окружающих его автомобилей и пешеходов, но и отслеживать дефекты на дорожном полотне, чтобы избежать резкого изменения траектории движения автомобиля или же его повреждения.

Все эти внешние факторы движения требуют от водителя внимания, отвлекая его от других важных моментов, с точки зрения безопасного дорожного движения. Поэтому ухудшение качества дорожного покрытия приводит к повышению аварийности.

Согласно информации ГИБДД, почти треть дорожно-транспортных происшествий в России происходит при сопутствующем влияния недостатков транспортно-эксплуатационного состояния дорог [2].

На рисунке 1 представлен график с показателями по авариям за 2021 год, на месте которых были обнаружены нарушения требований к состоянию дорог.

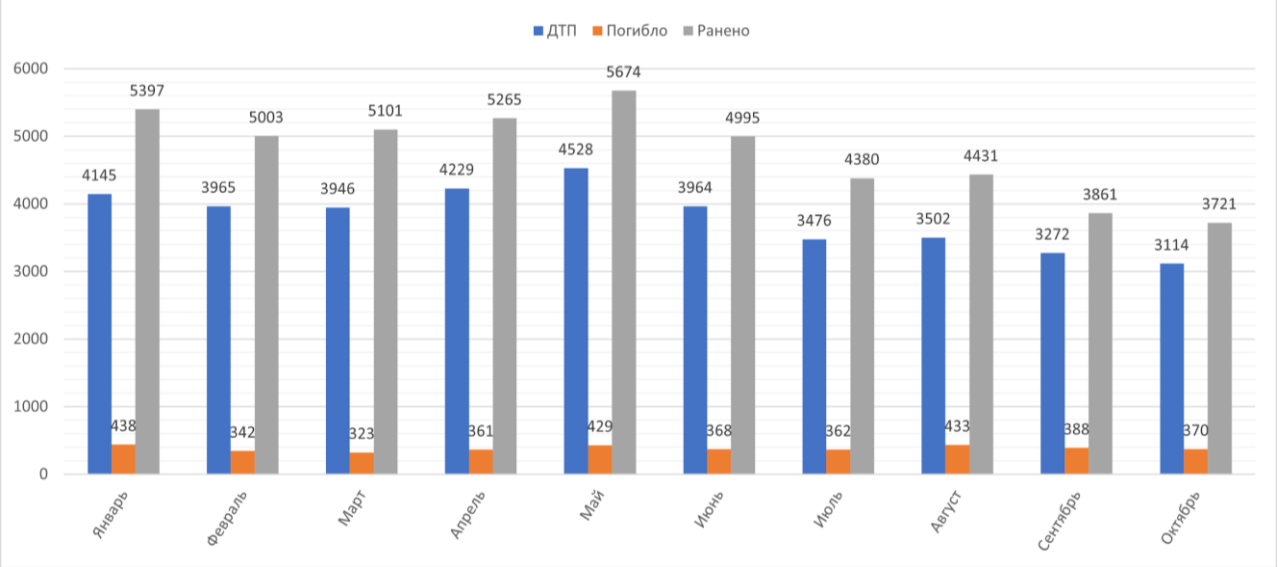


Рисунок 1. Показатели аварийности на дорогах с нарушением требований к их эксплуатационному состоянию.

Таким образом за 2021 год в России было выявлено более 38 тысяч ДТП, на месте которых были зафиксированы нарушения обязательных требований к состоянию автомобильных дорог [3]. В результате данных аварий было ранено более 38 тысяч человек, а количество погибших близилось к 4 тысячам.

На сегодняшний день существуют специальные службы, задачами которых является обеспечение надлежащего состояния автомобильных дорог. К главным задачам таких служб относится проведение плановых осмотров, выполнение диагностик, ремонт и прокладка нового дорожного полотна.

Проведение качественного анализа состояния дорожного покрытия и его ремонта невозможно без специальных устройств, которые позволяют выполнять не только диагностику ям на дорогах, но и выявлять на ранней стадии места, где начали появляться сколы, трещины, на месте которых может начаться процесс разрушения дорожного покрытия.

Исходя из того, что действующее законодательство РФ позволяет владельцу автомобиля получить компенсацию ущерба, причиненного его автомобилю в результате аварии по причине ненадлежащего состояния дорожного покрытия автомобильной дороги, можно сделать вывод, что данная проблема действительно актуальна.

Развитие машинного обучения способствовало распространению технологии компьютерного зрения. Эта технология позволяет роботам с помощью специальных алгоритмов и программ анализировать поток данных, который они получают с видеокамеры [4, с. 5].

Обнаружение объекта на изображении - задача установления соответствия яркостных областей заданного объекта с яркостными областями на изображении. В случае обнаружения объекта далее определяются его координаты на плоскости изображения.

Составление функции яркости изображения с заданным объектом является основным принципом обнаружения объектов. При реализации процедуры обнаружения полученный фрагмент яркости искомого объекта проверяется на соответствие с последовательно меняющейся областью на изображении. Каждое положение обрабатываемого поля проверяется на сходство с функцией яркости фрагмента. Ввиду того, что функция яркости лишь приближенно описывает искомый объект, точного совпадения заданного объекта с обнаруженным маловероятно [5, с. 4].

Нейросетевые технологии являются одним из основных методов, развиваемых в рамках искусственного интеллекта. Искусственные нейронные сети представляют собой программную реализацию математических моделей, которые моделируют работу биологических нейронных сетей, нервных клеток живого организма [6, с. 94].

Понятие нейронной сети возникло изучении и попытке моделирования процессов, протекающих в мозге. Первыми результатами данной работы являются нейронные сети У. Маккалока и У. Питтса [7].

Корневое отличие способа обработки информации человеческим мозгом от методов, которые применяются обычными цифровыми компьютерами, способствовало развитию исследований по искусственным нейронным сетям [8].

Человеческий мозг способен организовывать свои структурные компоненты, называемые нейронами, работа которых, в свою очередь,

обеспечивает молниеносное решение таких задач как обработка множества различных сигналов и распознавание образов. Примером обработки таких задач может служить обычное зрение [9, 10, 11].

В качестве другого примера можно рассмотреть активную эко-локационную систему летучей мыши, которая способна предоставлять информацию о расстоянии и относительной скорости нужного объекта, его размерах и высоте местоположения [12, 13].

Искусственные нейронные сети моделируют работу человеческого мозга, его нервной системы. Нервные волокна, связывающие между собой нейроны, способны отправлять электрические импульсы. Все процессы, связанные с получением информации, ее обработкой и интерпретацией, все это реализуется в живом организме как передача электрических импульсов между нейронами [14].

В отличии от методов, построенных на использовании традиционных алгоритмов, нейронные сети способны обучаться. Процесс обучения позволяет нейронным сетям автоматически выявлять и обобщать сложные зависимости между исследуемыми объектами и выходными данными. Это означает, что в процессе обучения сети учатся выдавать верный результат не только для тех данных, которые использовались при обучении, но и для тех, которые имеют сходные признаки с искомыми объектами (к примеру, зашумленные или частично искаженные данные) [15].

Особое место в решении задач компьютерного зрения занимают сверточные нейронные сети, нацеленные на эффективное распознавании образов на изображениях за счет применения операций свертки и пулинга, подробное описание которых будет представлено в главе 2.

Актуальность исследования поставленной проблемы определяется недостаточной разработкой и практическим применением методов и технологий компьютерного зрения, позволяющих выполнять обнаружение неровностей на дорогах.

Идея проекта заключается в разработке решения, основанного на применении искусственных нейронных сетей, которое позволит анализировать состояние проезжей части и выполнять мониторинг обнаруженных неровностей с их отображением на карте.

Основной целью работы является исследование и программная реализация методов глубокого обучения, позволяющих выполнять обнаружение ям на дорогах с точностью не менее 80%.

В рамках проекта были поставлены следующие задачи:

- анализ существующих технических и технологических решений;
- сбор и обработка данных;
- разметка данных;
- исследование архитектур нейронных сетей для проведения эксперимента;
- проведение вычислительного эксперимента для дальнейшего получения модели нейронной сети, отвечающей заранее заданным критериям;
- разработка приложения для обнаружения ям на дорожном покрытии;
- разработка приложения для мониторинга ям с их отображением на карте.

ГЛАВА 1. ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. АНАЛИЗ ТЕХНИЧЕСКИХ И ТЕХНОЛОГИЧЕСКИХ РЕШЕНИЙ

В ходе анализа технических и технологических решений было выявлено несколько методов, использование которых позволит выполнять обнаружение ям на дорогах:

- методы, основанные на измерении ускорения (применение акселерометра);
- методы, основанные на 3D-построении (применение лазерного сканирования);
- методы, основанные на компьютерном зрении (применение нейронных сетей).

Применение методов, основанных на измерении ускорения, были рассмотрены в работах Д. С. Людовских [17], B. Lanjewar [18], M. Hoffmann [19] и J. Eriksson [20]. Использование данного метода эффективно только тогда, когда водитель преднамеренно двигается по ямам на дорогах. Это обусловлено тем, что акселерометр измеряет ускорение по осям X (вперед, назад), Y (влево, вправо), Z (вверх, вниз). Обнаружение неровности возможно только измерением ускорения по оси Z, потому что уклонение от ямы по оси Y нельзя считать за попытку объехать неровность на дороге, поскольку чаще всего такие маневры выполняют для совершения обгона или перестроения. Ввиду того, что ямы имеют разные размеры, возможны случаи, когда значение ускорения по оси Z при попадании автомобиля в яму может быть таким же, как и при его попадании в дорожный разлом, колею или при его движении по искусственной неровности (к примеру, лежачий полицейский). Именно поэтому данные с акселерометра не позволяют точно выполнять классификацию обнаруженных неровностей.

Методы, основанные на применении лазерного сканирования, являются самыми точными. Обнаружение ям на дорогах при помощи лидара были рассмотрены в работах V. Prasad [21], B. Kang [22], R. Ravi [23] и K. Chang [24].

Использование данного метода позволяет не только выполнить обнаружение неровности, но и определить ее точные размеры и позицию относительно текущих координат автомобиля, но при этом требует соответствующего дорогостоящего оборудования.

Методы, основанные на зрении, используют различные способы обработки изображений [25], [26] для обнаружения ям на 2D-изображениях или видеоданных. Именно при помощи искусственных нейронных сетей, можно получить оптимальный результат при небольших затратах в отличии от использования других методов.

Эффективность применения нейронных сетей для обнаружения неровностей на дорогах была подтверждена результатами работ S. Ansari [27], A. Aparna [28] и R. Fan [29]. Так, например, A. Aparna в своей работе выполнил обучение модели ResNet101 на наборе данных, состоящем из 4320 изображений, из которых 10% составляла валидационная выборка. Точность на валидационных данных обученной модели обнаружения ям составила 92,5%.

1.2. ТРЕБОВАНИЯ К СОСТОЯНИЮ ДОРОЖНОГО ПОКРЫТИЯ.

В России существуют различные категории дорог и улиц, представленные в Приложении 1, к состоянию которых в зависимости от группы улиц предъявляются определенные требования, представленные в Приложении 2.

В рамках данной работы будут рассмотрены одни из наиболее опасных повреждений в виде ям на дороге, наличие которых регулируется соответствующим государственным стандартом - ГОСТ Р 50597–2017 [16].

.

ГЛАВА 2. ОСНОВНЫЕ ВИДЫ АРХИТЕКТУР НЕЙРОННЫХ СЕТЕЙ

Новые виды архитектур нейронных сетей появляются постоянно. В данной главе будут представлены основные виды архитектур с кратким описанием, а также рассмотрен современный алгоритм глубокого обучения - YOLO.

2.1. НЕЙРОННЫЕ СЕТИ ПРЯМОГО РАСПРОСТРАНЕНИЯ И ПЕРЦЕПТРОНЫ

Нейронные сети прямого распространения (feed forward neural networks, FF или FFNN) и перцептроны (perceptrons, P) представляют собой прямолинейные сети (см. рисунок 2), в которых передача информации между нейронами осуществляется по направлению от входа к выходу.

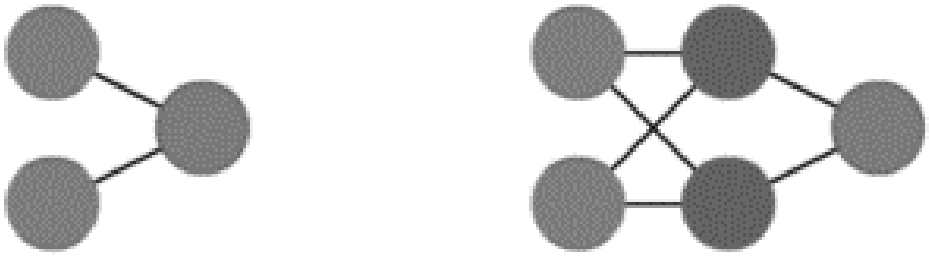


Рисунок 2. Нейронные сети прямого распространения и перцептрон.

Самая простая нейронная сеть имеет две входных клетки и одну выходную. Такие сети обычно обучаются по методу обратного распространения ошибки, в котором сеть получает множества входных и выходных данных. Обучение методом обратного распространения ошибки представляет собой нахождение разницы между вводом и выводом [30].

Такой процесс обучения нейросети является обучением с учителем потому, что в данном случае множество выходных данных составляется не автоматически.

Как правило такие сети комбинируют с другими видами сетей для получения новых.

2.2. СВЁРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ

Свёрточные нейронные сети (convolutional neural networks, CNN) и глубинные свёрточные нейронные сети (deep convolutional neural networks,

DCNN) в некоторой степени моделируют зрительную кору головного мозга. Небольшие участки клеток нейронов, расположенных на зрительной коре, связаны с определенными местами зрительного поля. В 1962 году был проведен эксперимент, в котором подтвердили, что некоторые клетки нейронов активируются при наблюдении границ конкретной ориентации. Хьюбел и Визель определили, что все эти нейроны сконцентрированы в образе стержневой построения и основывают визуальное восприятие человека [31].

Сверточные нейронные сети эффективны для обработки визуальных и других двумерных данных. CNN может состоять из нескольких сверточных слоев с полностью соединенными нейронами. В отличии от других глубоких архитектур, сверточные нейронные сети показали отличные результаты в обработке графических данных. CNN подобно прямолинейным сетям могут обучаться методом обратного распространения ошибки. При этом сверточные нейронные сети обучаются быстрее и имеют меньше параметров, что делает их весьма привлекательными [32]. На рисунке 3 представлен пример сверточной нейронной сети.

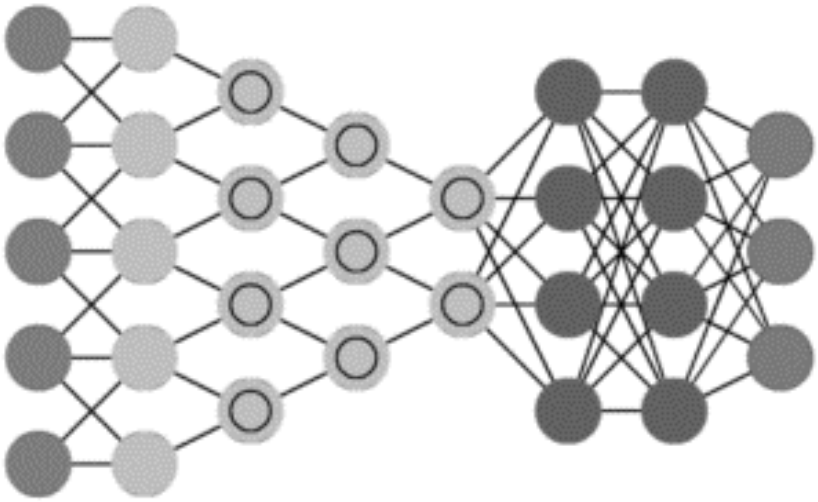


Рисунок 3. Сверточная нейронная сеть.

Сверточные нейронные сети как правило состоят из нескольких слоев: сверточный слой, слой объединения и полносвязный слой. Обобщенная архитектура сверточной нейронной сети представлена на рисунке 4.

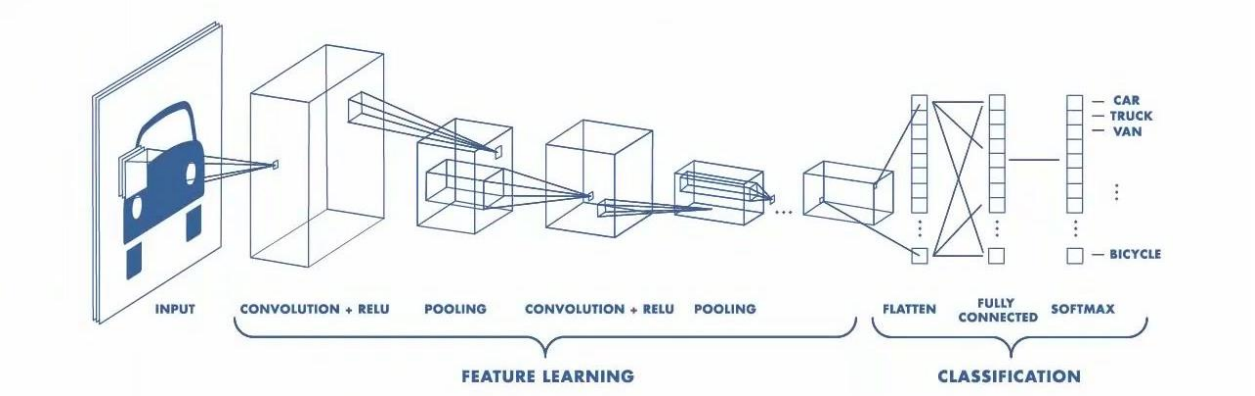


Рисунок 4. Обобщенная архитектура сверточной нейронной сети [33].

Сверточный слой является основным слоем данной архитектуры. На этом уровне выполняется перемножение матрицы изображения и ядра (набора обучаемых параметров).

Во время прямого прохода ядро выполняет вычисления по высоте и ширине изображения. Получаемое двумерное представление изображения называется активацией (Activation). Скользящий размер матрицы набора обучаемых параметров называется шагом.

Размер выходного слоя можно определить по формуле 1:

$$Wout = \frac{W-F+2P}{S}, \qquad (1)$$

где W - размер входного слоя, F - пространственный размер, P - число отступов, S - шаг. При этом выходное разрешение полученного слоя будет равняться W_{out} * W_{out} * D_{out}.

При помощи слоя пулинга можно сократить пространственное представление изображения и тем самым увеличить скорость вычислений в нейронной сети. Сокращение пространственного представления выполняется за счет применения функции объединения. Наиболее популярной функцией является максимальный пулинг, когда из набора близлежащих значений матрицы выбирается максимальное значение.

Размер выходного слоя после операции пулинга высчитывается по следующей формуле 2:

$$Wout = \frac{W-F}{S} + 1, \qquad (2)$$

где W - размер входного слоя, F - пространственный размер, S - шаг.

Используя комбинацию двух данных слоев, мы уменьшаем исходные размеры изображения. Многократное применение данной комбинации способствует созданию более глубоких сверточных нейронных сетей.

Прогнозирование выходных данных выполняется на полносвязном слое с применением функции-активатора: при бинарном выходе - сигмоидная, иначе - многопеременная функция. Полносвязный слой генерирует N-пространственный вектор для входных данных, где N – количество классов, на которых была обучена сверточная нейронная сеть.

Работа полносвязного слоя основана на обращении к результатам работы предыдущего слоя и вычислении характеристик, связанных со свойствами определенных классов. Например, для программы распознавания кошек и собак результирующий вектор полносвязного слоя будет состоять из набора вероятностей (двух элементов) – [0.2, 0.8]. Таким образом, полносвязный слой говорит о том, что существует 20% вероятность, что на изображении обнаружен кот или же 80% того, что обнаружена собака.

2.4. НЕЙРОННЫЕ СЕТИ С АРХИТЕКТУРОЙ YOU ONLY LOOK ONCE

Существуют множество алгоритмов обнаружения объектов, такие как Single Shot Detector (SSD), Histogram of Oriented Gradients (HOG), Faster R-CNN, YOLO (You Only Look Once) и т. д.

Одним из современных алгоритмов обнаружения объектов является алгоритм, связанный с архитектурой YOLO, главным отличием от других алгоритмов глубокого обучения которого является то, что он способен выполнять обнаружение объектов в режиме реального времени.

Основной принцип данного алгоритма предусматривает обработку всего изображения сверточной нейронной сетью только один раз. Другие алгоритмы выполняются данный процесс многократно. Модели YOLO способны усваивать расположение объектов на изображении и их относительные размеры [34].

Идея данного алгоритма предполагаете разбиение сетки изображения на ячейки размером X на X, каждая из которых выполняет предсказывание содержащих ограничивающих рамок и их классов.

Скорость работы алгоритма YOLO обусловлена тем, что он рассматривает задачу обнаружения объекта на изображении как регрессионную задачу. В отличии от алгоритмов, основанных на скользящем окне, YOLO рассматривает все изображение во время обучения и тестирования, тем самым выделяя информацию об объектах и их внешнем виде.

Рассматривание более широкого контекста при обработке изображения позволяет алгоритму увидеть внешние признаки, характеризующие наличие или отсутствие объекта. Именно поэтому у данных сетей больше шансов предсказать наличие необходимого объекта на зашумленном изображении.

ГЛАВА 3. ИССЛЕДОВАНИЕ ВЛИЯНИЯ АРХИТЕКТУР НЕЙРОННЫХ СЕТЕЙ НА ТОЧНОСТЬ ОБНАРУЖЕНИЯ ЯМ

В данной главе будут описаны этапы проведения эксперимента по исследованию влияния различных моделей нейронных сетей на точность обнаружения ям.

Ввиду того, что процесс обучения моделей будет ограничен определенным техническими ресурсами, представленными в Приложении 3, модели с наибольшим количеством параметров (связей между нейронами) будут обучаться с как можно большими значениями параметров обучения (размер обучающей выборки, размер входного изображения), модели с наименьшим количеством параметров – наоборот.

3.1. ОПРЕДЕЛЕНИЕ МОДЕЛЕЙ ДЛЯ ПРОВЕДЕНИЯ ЭКСПЕРИМЕНТА

Поскольку для задач классификации и обнаружения объектов предназначены сверточные нейронные сети, в эксперименте будут использованы модели Scaled YOLOv4 [35] и YOLOv5 [36] различных вариаций.

Модели с архитектурой Scaled YOLO v4 являются одними из лучших нейронных сетей для обнаружения объектов, средняя точность самой тяжелой модели на наборе данных MS COCO достигает 55.8%. На рисунке 5 представлены результаты тестирования модели Scaled YOLOv4 относительно других моделей.

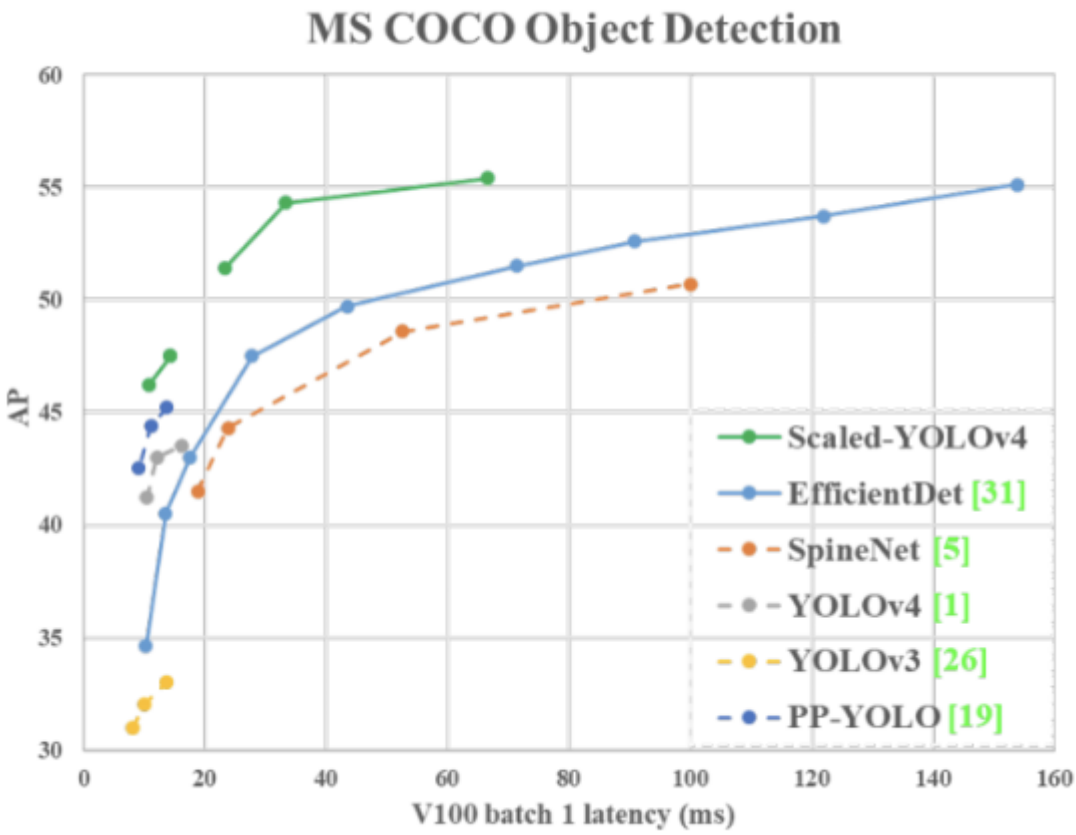


Рисунок 5. Средняя точность модели Scaled-YOLOv4 на наборе данных MS COCO [35].

Традиционный метод масштабирования модели Scaled YOLOv4 заключается в изменении глубины модели, то есть в добавлении большего количества сверточных слоев [37]. Архитектура модели Scaled YOLOv4-large, в том числе YOLOv4-P5, YOLOv4-P6 иYOLOv4-P7 представлена на рисунке 6.

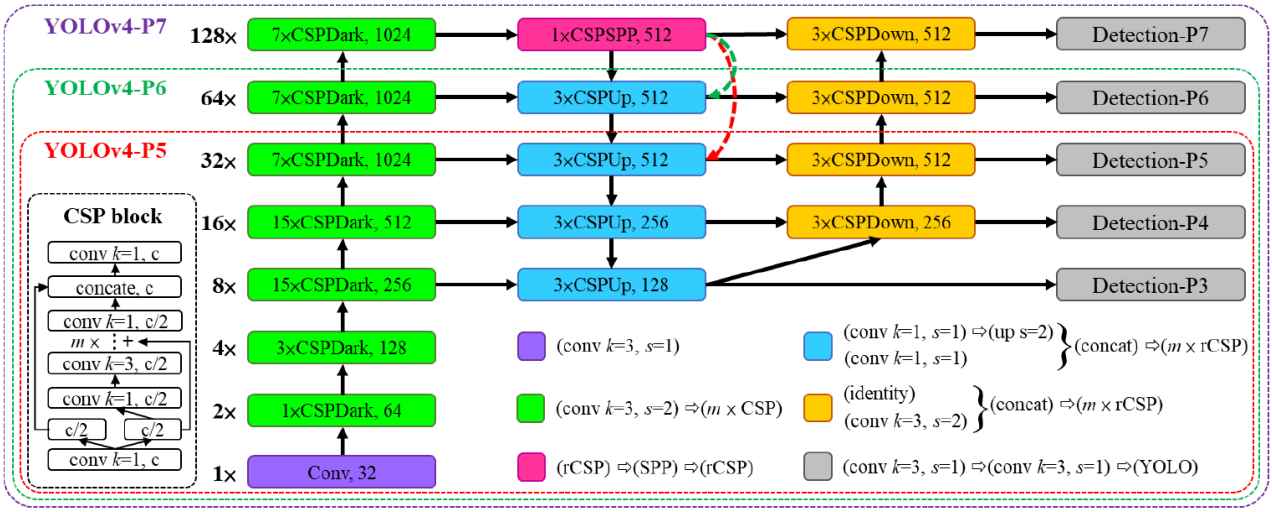


Рисунок 6. Архитектура модели Scaled YOLOv4-large [37].

При помощи CSP-соединений выполняется разделение выходного сигнала, часть из которого движется по основному пути, отвечающем за генерацию семантической информации с большим рецептивным полем, часть - по обходному пути, сохраняя пространственную информацию.

YOLOv5 – это новейшая модель обнаружения объектов, разработанная ultralytics той же компанией, которая разработала PyTorch-версию YOLOv3 и была выпущена в июне 2020 года [38].

Модель YOLOv5 доступна в четырех вариациях – small, medium, large, x-large, каждая из которых показывает свою точность при фиксированной производительности (см. рисунок 7).

Разница между вариациями моделей YOLOv5 заключается в количестве имеющихся параметров (связей между нейронами). Так модель YOLOv5 вариации small имеет 7,2 миллиона параметров, а вариация x-large – состоит из 86,7 миллионов параметров.

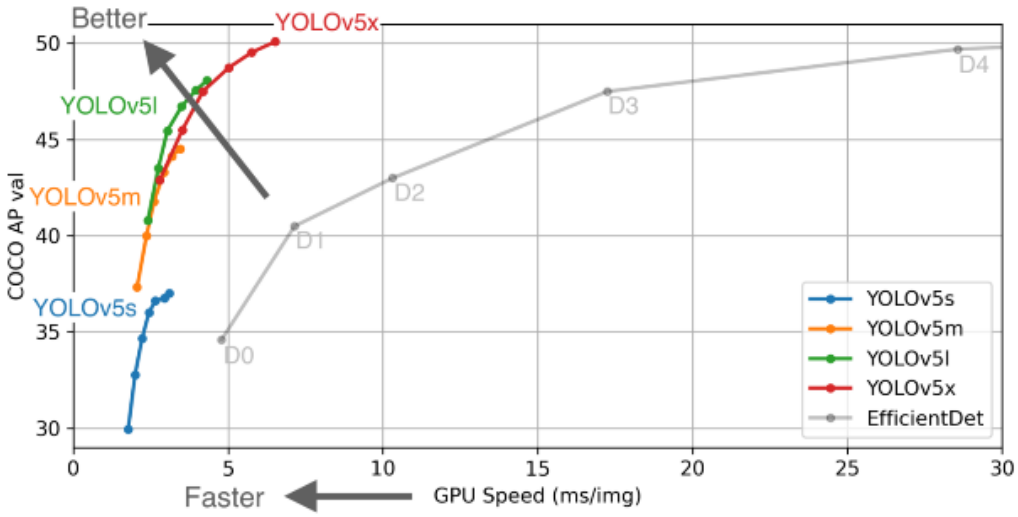


Рисунок 7. Средняя точность моделей YOLOv5 на наборе данных MS COCO [36].

YOLOv5 имеет архитектуру One-Stage detector (см. рисунок 8). Одноступенчатое обнаружение в отличии от многоступенчатых детекторов позволяет сократить скорость вычисления ограничивающих областей – данный подход сразу же предсказывает все возможные координаты определённого количества объектов с их вероятностью, и в дальнейшем корректирует их местоположение.

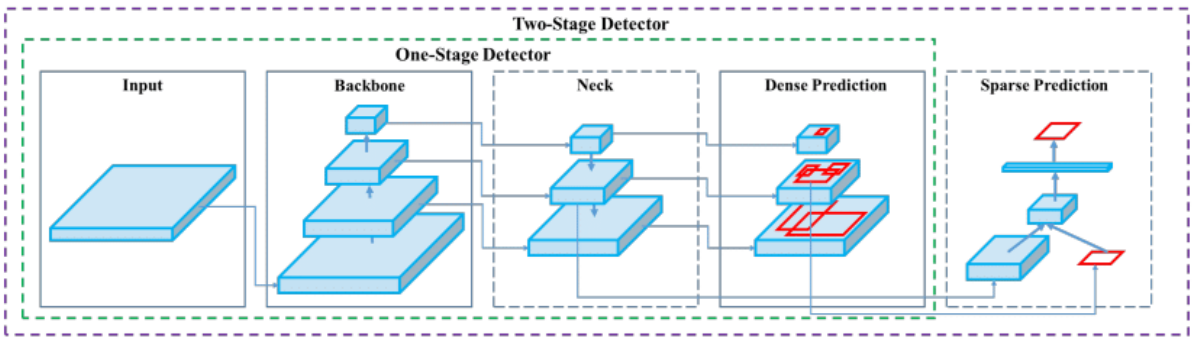


Рисунок 8. One-Stage detector [39].

На основании описанных выше моделей Scaled-YOLOv4 и YOLOv5 был составлен список моделей, используемых для обучения, которые представлены в таблице 1. Ввиду ограниченной производительности вычислительных ресурсов модели с большим количество связей между нейронами будут тренироваться с меньшим размером обучающих выборок и размером изображений. Все модели будут обучаться 300 эпох.

Таблица 1

Список моделей и параметров их обучения.

Модель	Размер изображений (image size)	Размер обучающих партий (batch size)	Количество эпох
YOLOv5n6	1280px	64	300
YOLOv5s	640px	128	
YOLOv5s6	1280px	32	
YOLOv5m	640px	64	
YOLOv5l	640px	64	
YOLOv5l6	1280px	16	
YOLOv5x	640px	64	
YOLOv5x6	1280px	16	
Scaled-YOLOv4-P6	1280px	16	

3.2. ПОДГОТОВКА И РАЗМЕТКА ДАННЫХ ДЛЯ ОБУЧЕНИЯ МОДЕЛЕЙ

В работе использовался набор данных о ямах, составленный на факультете электротехники и электроники Стелленбосского университета для проведений исследования по обнаружению неровностей на дорожном покрытии [40], [41].

Набор данных состоит из двух разных выборок, одна из которых считается простой, а другая более сложной для распознавания. Эти выборки могут иметь общие образцы, и есть несколько случаев, когда два разных изображения будут имеют одно и то же имя. Поэтому на этапе обработки

исходных данных все изображения были помещены в один каталог, отсортированы по размеру с последующим удалением дубликатов.

Набор данных представляет собой снимки с разрешением 3680 на 2760 пикселей, сделанные из салона автомобиля (см. рисунок 9). Общее количество изображений в наборе данных – 706.

Рисунок 9. Примеры изображений из набора.

Ввиду того, что на этапе определения моделей для проведения исследования были выбраны модели, которые имеют разную архитектуру и требуют различные форматы файлов графической аннотации, разметка данных осуществлялась при помощи облачного инструмента Roboflow [42], который позволяет выполнять разметку данных (см. рисунок 10) в едином формате с дальнейшей возможностью его преобразования в необходимый для обучения модели формат.

Рисунок 10. Roboflow – разметка данных.

После разметки данных при помощи данного инструмента был сгенерирован набор данных для обучения. Генерация набора данных осуществлялась путем разделения датасета на обучающую, валидационную и тестовую выборку (см. рисунок 11).

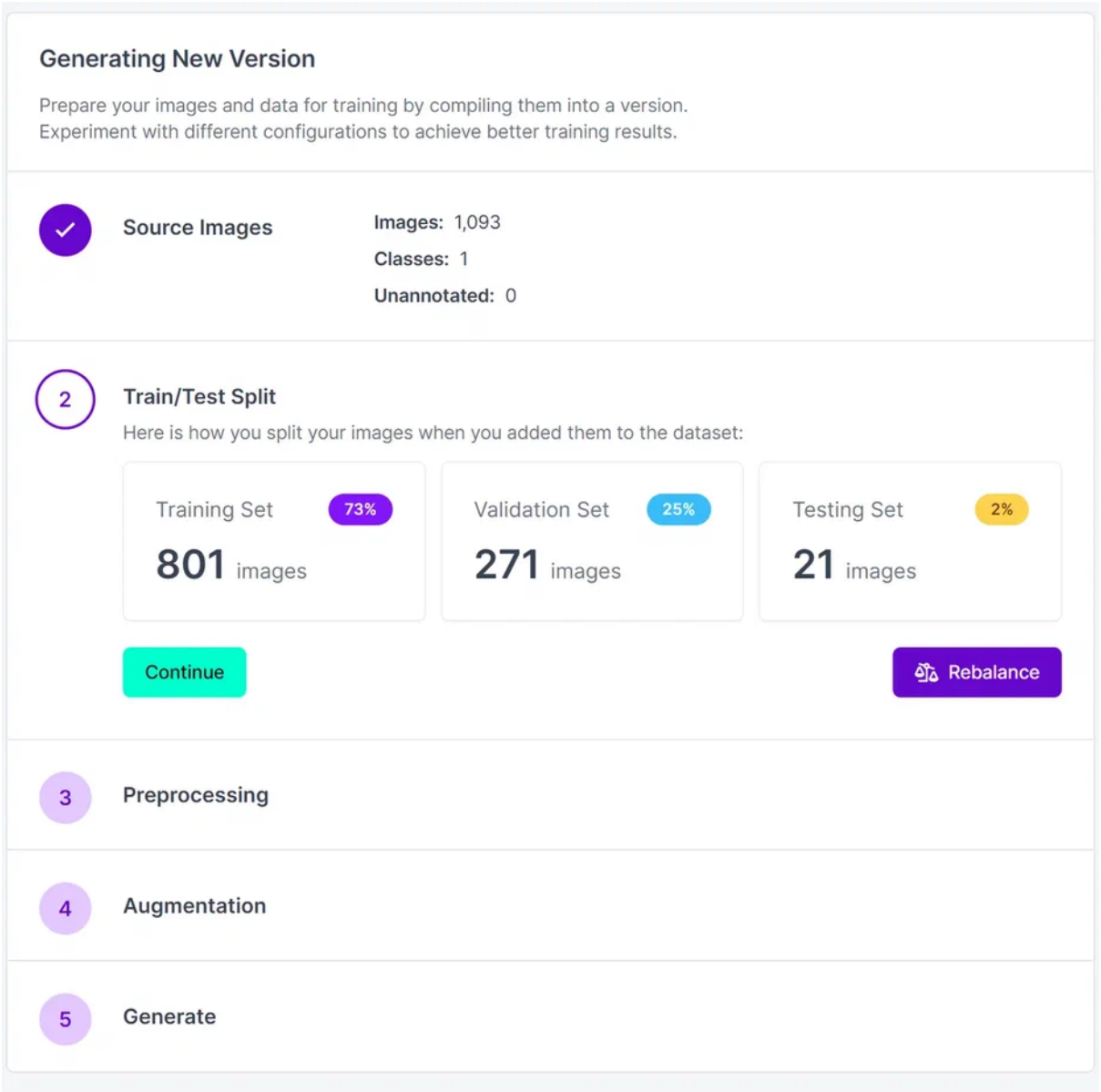


Рисунок 11. Roboflow – разделение на выборки.

Далее сервис предлагает выполнить предварительную обработку данных (см. рисунок 12), можно выполнить такие операции как: изменение размеров изображение, обрезка изображений с изоляцией размеченных объектов, преобразование изображения в серые оттенки и некоторые другие.

Рисунок 12. Roboflow – предварительная обработка данных.

После предварительной обработки данных Roboflow предлагает выполнить аугментацию – расширение набора данных за счет создания (см. рисунок 13).

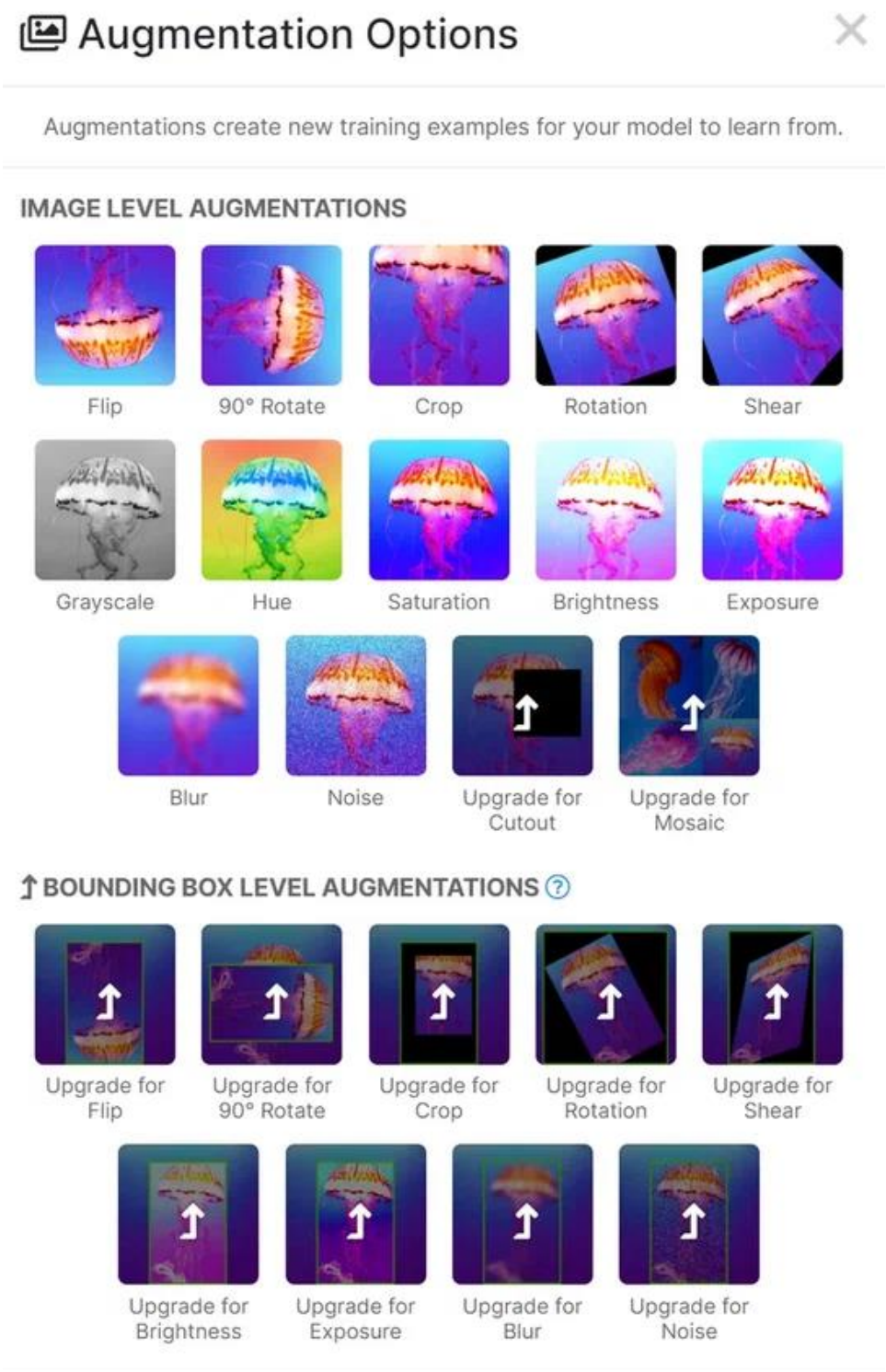


Рисунок 13. Roboflow – аугментация данных.

Расширение набора данных выполняется за счет применения фильтров как к исходным изображениям, так и к размеченным объектам на изображениях. Сервис позволяет выполнить следующие операции с изображениями для

аугментации данных: отражение, вращение, обрезка, сдвиг, изменение цветового режима, яркости, зашумления и другое.

Сгенерированный мной набор данных для обучения состоит из 1093 снимков с разрешением не менее 1280x720 пикселей, где обучающая выборка состоит из 801 изображения, валидационная – из 271 изображения и тестовая – из 21 изображения.

Для загрузки полученного набора данных нужно нажать на кнопку «Export» и выбрать необходимый формат. Сервис поддерживает все имеющиеся форматы разметки данных (см. рисунок 14).

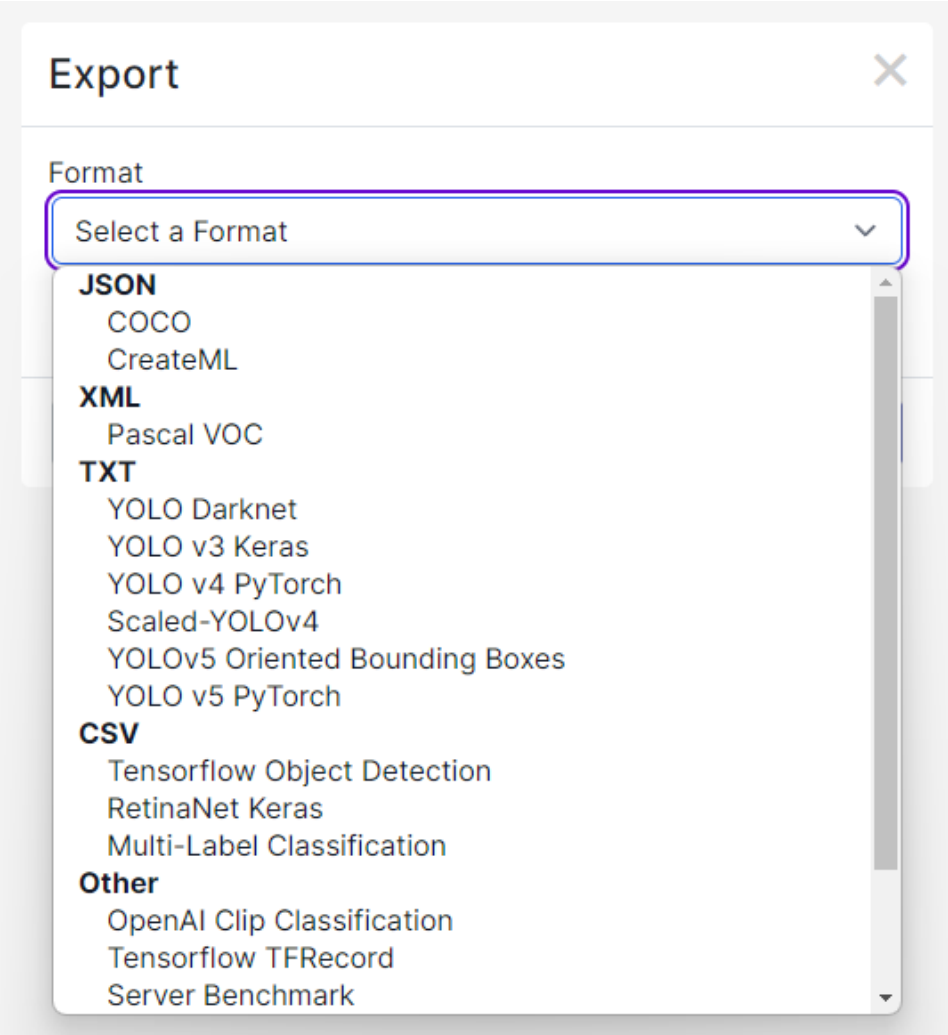


Рисунок 14. Roboflow – экспорт данных.

В моем случае экспорт набора данных осуществлялся в форматах Scaled-YOLOv4 и YOLOv5 PyTorch.

3.3. ОБУЧЕНИЕ МОДЕЛЕЙ

Обучение моделей осуществлялось методом трансферного обучения. Идея трансферного обучения заключается в применении предварительно обученных сетей с целью сокращения времени обучения и повышения точности обучения модели на новых данных. Это связано с тем, что параметры сети уже настроены и обучения всей сети с нуля не выполняется, что значительно ускоряет процесс развертывания глубокой нейронной сети [43].

Трансферное обучение становится актуальным в области глубокого обучения из-за огромного количества вычислительных ресурсов, которые необходимы для работы с большими и сложными наборами данных.

Обучение моделей проходило на удаленном сервере с графическим процессором NVIDIA Tesla V100 с 32 гигабайтами видеопамяти. Ввиду того, что процесс обучения ограничен мощностями, определенным в Приложении 3, модели нейронных сетей, требующие высокой производительности, тренировались с меньшим размером обучающих партий, а модели, обучение которых занимало меньшее количество ресурсов – с наибольшим размером обучающих партий.

Таким образом, в одном случае мы повышаем точность за счет увеличения размеров обучающих выборок, в другом – за счет увеличения количества связей в нейронной сети.

Количество эпох, используемых при обучении всех моделей, равнялось 300. В качестве инструмента машинного обучения использовался PyTorch. В качестве параметров обучения для каждой модели задавались соответствующие ей параметры из таблицы 1.

С целью отслеживания процесса обучения применялся инструмент WandB [44], который позволяет просматривать гиперпараметры, системные показали и прогнозы в режиме реального времени. Интерфейс инструмента представлен на рисунке 15.

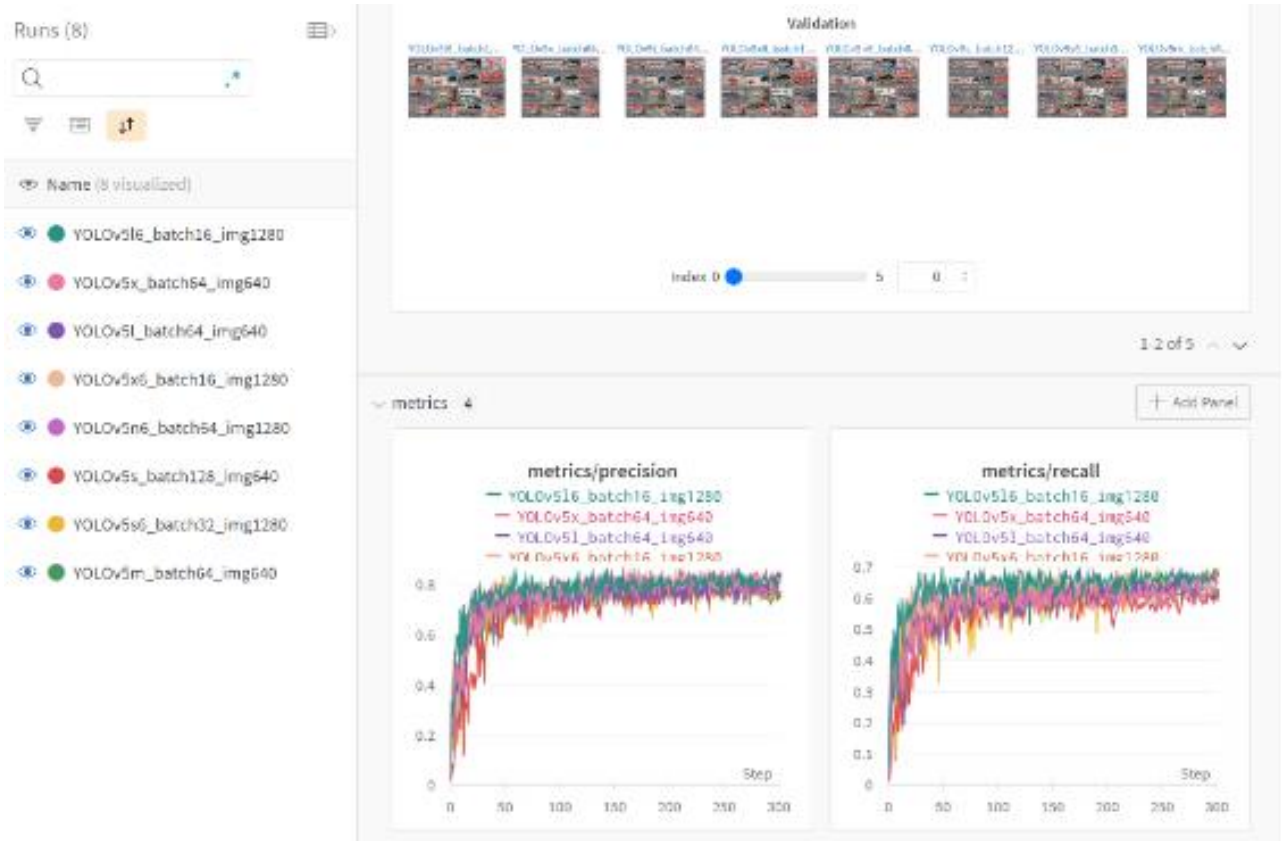


Рисунок 15. WandB – интерфейс инструмента.

Результаты обучения моделей представлены на рисунке 16. В Приложении 4 представлены результаты обучения моделей с ID 1-4, в Приложении 5 – результаты обучения моделей с ID 5-8. Модель Scaled-YOLOv4 продемонстрировала наименьшую точность, результаты ее обучения указаны в Приложении 6.

Name (8 visualized)	ID	epochs	best/epoch	best/mAP_0.5	best/mAP_0.5:0.95	best/precision	best/recall
YOLOv5l6_batch16_img1280	1	300	260	0.769	0.4634	0.8429	0.677
YOLOv5x_batch64_img640	2	300	292	0.7185	0.4212	0.8398	0.6005
YOLOv5l_batch64_img640	3	300	268	0.7188	0.4203	0.8325	0.6097
YOLOv5x6_batch16_img1280	4	300	293	0.7371	0.4505	0.8099	0.6557
YOLOv5n6_batch64_img1280	5	300	246	0.7269	0.4089	0.7647	0.6734
YOLOv5s_batch128_img640	6	300	275	0.6864	0.3774	0.7615	0.6181
YOLOv5s6_batch32_img1280	7	300	298	0.7437	0.4318	0.7559	0.6898
YOLOv5m_batch64_img640	8	300	291	0.7039	0.4149	0.7468	0.6285

Рисунок 16. WandB – результаты обучения моделей.

В ходе эксперимента с учетом доступных вычислительных мощностей, и следовательно, с учетом параметров, используемых при обучении, наибольшую точность продемонстрировала модель YOLOv5l6, на достижение наибольшей точности, равной 0.8429, ей потребовалось 260 эпох обучения. На рисунке 17 представлен скриншот файла-журнала, показывающий информацию о данной модели.

```
                from  n    params  module                                  arguments
  0                -1  1      7040  models.common.Conv                      [3, 64, 6, 2, 2]
  1                -1  1     73984  models.common.Conv                      [64, 128, 3, 2]
  2                -1  3    156928  models.common.C3                        [128, 128, 3]
  3                -1  1    295424  models.common.Conv                      [128, 256, 3, 2]
  4                -1  6   1118208  models.common.C3                        [256, 256, 6]
  5                -1  1   1180672  models.common.Conv                      [256, 512, 3, 2]
  6                -1  9   6433792  models.common.C3                        [512, 512, 9]
  7                -1  1   3540480  models.common.Conv                      [512, 768, 3, 2]
  8                -1  3   5611008  models.common.C3                        [768, 768, 3]
  9                -1  1   7079936  models.common.Conv                      [768, 1024, 3, 2]
 10                -1  3   9971712  models.common.C3                        [1024, 1024, 3]
 11                -1  1   2624512  models.common.SPPF                      [1024, 1024, 5]
 12                -1  1    787968  models.common.Conv                      [1024, 768, 1, 1]
 13                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 14          [-1, 8]  1         0  models.common.Concat                    [1]
 15                -1  3   6200832  models.common.C3                        [1536, 768, 3, False]
 16                -1  1    394240  models.common.Conv                      [768, 512, 1, 1]
 17                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 18          [-1, 6]  1         0  models.common.Concat                    [1]
 19                -1  3   2757632  models.common.C3                        [1024, 512, 3, False]
 20                -1  1    131584  models.common.Conv                      [512, 256, 1, 1]
 21                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 22          [-1, 4]  1         0  models.common.Concat                    [1]
 23                -1  3    690688  models.common.C3                        [512, 256, 3, False]
 24                -1  1    590336  models.common.Conv                      [256, 256, 3, 2]
 25         [-1, 20]  1         0  models.common.Concat                    [1]
 26                -1  3   2495488  models.common.C3                        [512, 512, 3, False]
 27                -1  1   2360320  models.common.Conv                      [512, 512, 3, 2]
 28         [-1, 16]  1         0  models.common.Concat                    [1]
 29                -1  3   5807616  models.common.C3                        [1024, 768, 3, False]
 30                -1  1   5309952  models.common.Conv                      [768, 768, 3, 2]
 31         [-1, 12]  1         0  models.common.Concat                    [1]
 32                -1  3  10496000  models.common.C3                        [1536, 1024, 3, False]
 33  [23, 26, 29, 32]  1     46152  models.yolo.Detect                      [1, [[19, 27, 44, 40, 38, 94], [..]]
Model summary: 607 layers, 76162504 parameters, 76162504 gradients, 110.2 GFLOPs
```

Рисунок 17. Вывод файла-журнала при обучении.

Несмотря на преимущества метода масштабирования модели ScaledYOLOv4, данная модель продемонстрировала наименьшую точность равную 0.56.

На тестовом наборе данных было проведено тестирование модели YOLOv5l6, результаты которого представлены на изображениях 18 и 19.

Рисунок 17. Результат обнаружения ям на дорогах моделью YOLOv5l6 №1.

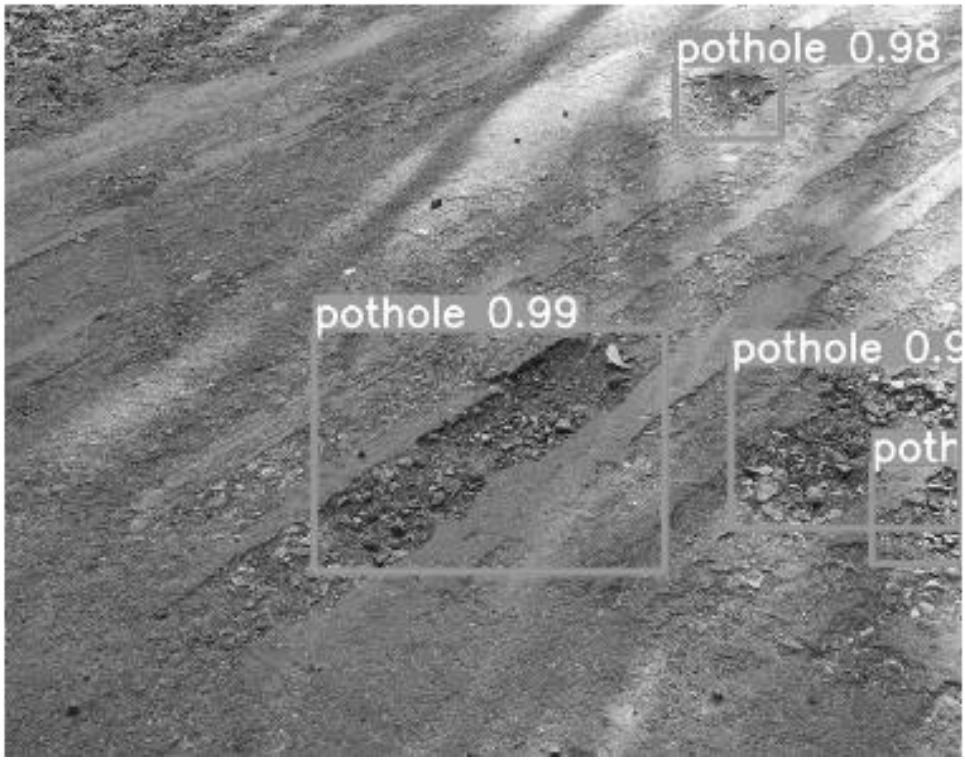


Рисунок 18. Результат обнаружения ям на дорогах моделью YOLOv5l6 №2.

ГЛАВА 4. РАЗРАБОТКА СИСТЕМЫ МОНИТОРИНГА ЯМ С ИХ ОТОБРАЖЕНИЕМ НА КАРТЕ

В рамках разработки системы мониторинга ям на карте было разработано приложение для обнаружения ям и сервис для визуализации полученных результатов с целью дальнейшего мониторинга состояния дорог с отображением ям на карте. Разработка осуществлялась с использование программного стека PERN – PostgreSQL (объектно-реляционная база данных), express (фреймворк для веб приложений), React (библиотека для пользовательских интерфейсов) и Node JS (программная платформа для взаимодействия с устройствами через API).

Архитектура разработанной системы представлена на рисунке 20.

Рисунок 20. Архитектура программно-аппаратной системы мониторинга неровностей дорожного покрытия.

Устройство обнаружения взаимодействует с сервисом мониторинга ям на карте при помощи API-запросов. В случае обнаружения ямы оно отправляет на сервис изображение с обнаруженными объектами и информацией о них и данные о географическом положении автомобиля.

Рассмотрим подробнее разработку каждой из частей.

4.1. РАЗРАБОТКА СЕРВИСА ДЛЯ МОНИТОРИНГА НЕРОВНОСТЕЙ

С целью мониторинга ям на карте был реализован сервис, который позволяет пользователям просматривать состояние дорог. Разработанный сервис состоит из двух частей – клиентской и серверной.

4.1.1. РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ ПРИЛОЖЕНИЯ

С использованием JavaScript-библиотеки React [45] была реализована клиентская часть приложения, структура которого представлена на рисунке 21.

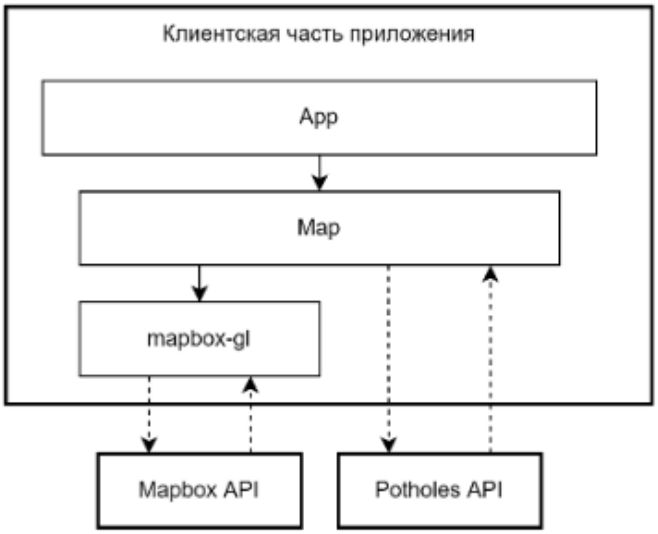


Рисунок 21. Структура клиентской части приложения.

Взаимодействие с картами происходит через API сервиса Mapbox [46], который предоставляет динамические, производительные и настраиваемые карты, соответствующие потребностям пользователей.

Загрузка набора данных о ямах на дорогах выполняется запросом к серверной части приложения через HTTP-клиент Axios. Основной компонент Map использует хуки React – useState и useEffect. Листинг кода данного компонента представлен в Приложении 7.

Хук useState необходим для объявления «переменной состояния». Это способ «сохранять» некоторые значения между вызовами функций. Обычно переменные теряются при выходе из функции, но переменные состояния сохраняются в React [47].

Используя useEffect, мы сообщаем React, что компонент должен что-то делать после отрисовки. React запомнит переданную и вызовет ее после обновления DOM [48]. В нашем случае мы извлекаем данные и вызываем API.

В зависимости от количества ям, расположенных в одной области, на карту добавляются маркеры с цветом, характеризующим состояние дорог (см. рисунки 22 и 23).

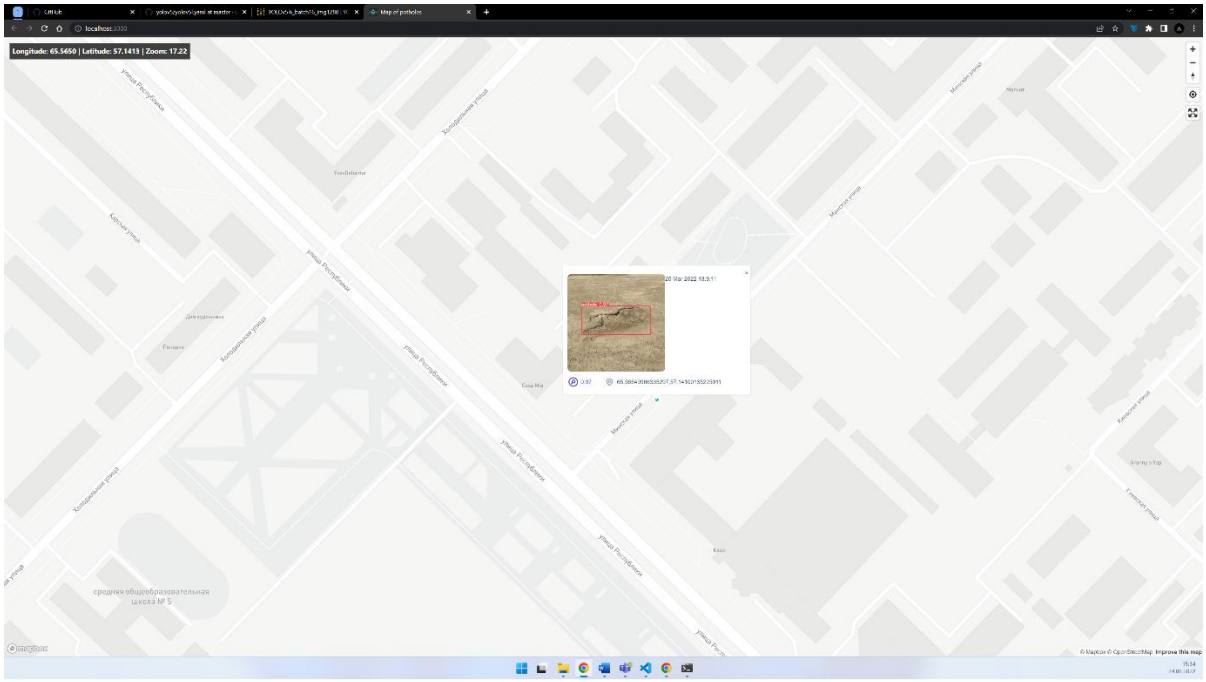


Рисунок 22. Прототип сервиса для мониторинга ям на карте (детализация дефекта).

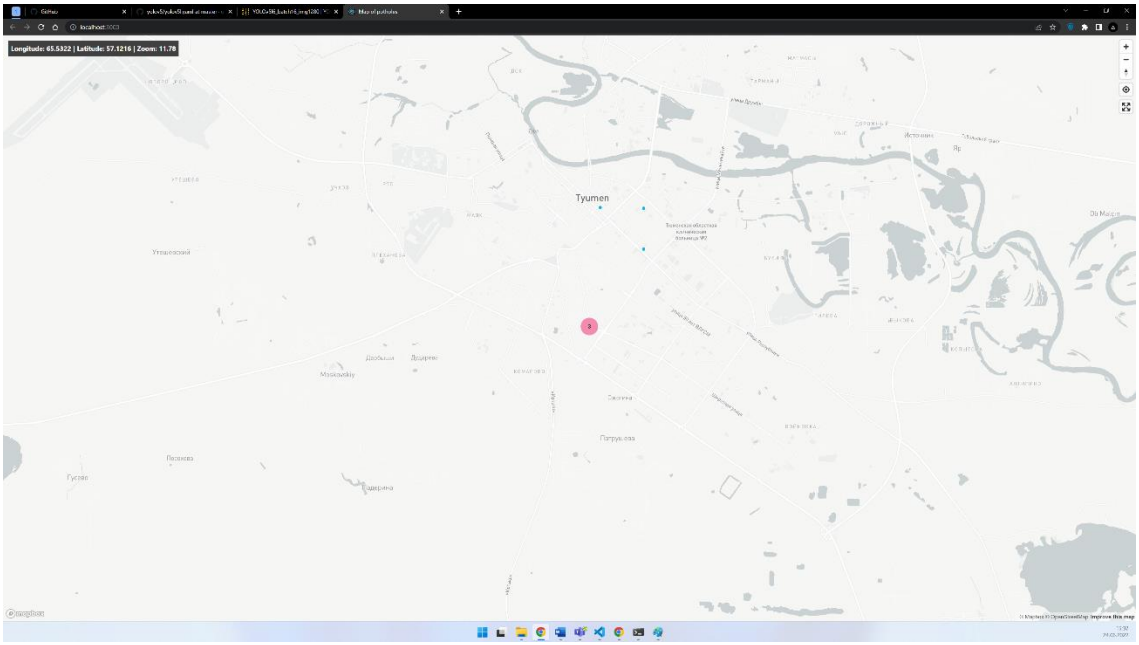


Рисунок 23. Прототип сервиса для мониторинга ям на карте (оценка состояния дорог).

4.1.2. РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ ПРИЛОЖЕНИЯ

Серверная часть приложения реализована при помощи программной платформы Node JS [49]. Создание API-маршрутов выполнялось в библиотеке express.

Ввиду того, что серверная часть должна была предоставлять возможность не только получать данные, при помощи библиотеки express-fileupload был разработан маршрут для загрузки снимков обнаруженных ям в заранее созданный каталог. При загрузке каждому снимку присваивается уникальный идентификатор.

Кроме загрузки снимков, серверная часть позволяет передавать географические координаты места, в котором произошло обнаружение, и точность, с которой нейросеть определила яму. На рисунке 24 представлена структура серверной части приложения.

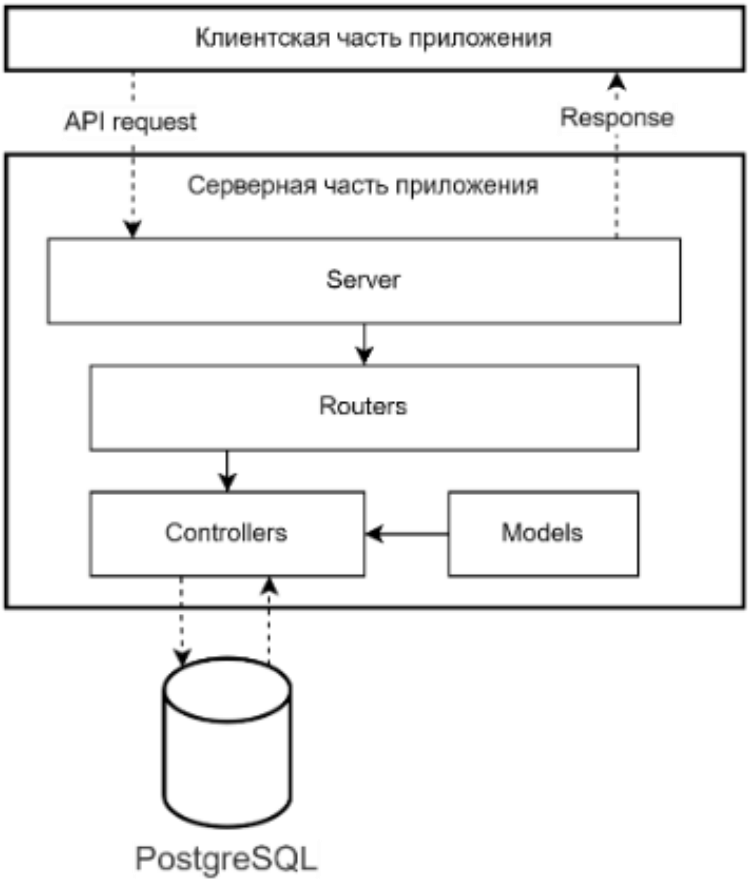


Рисунок 24. Архитектура серверной части приложения.

В качестве хранилища данных была выбрана объектно-реляционная система управления базами данных PostgreSQL. Настройка базы данных, хранилища и соединение с ней выполняется при помощи библиотеки Sequelize.

Инициализация таблиц в базе данных происходит на основании созданных моделей, в моем случае, набор моделей состоит всего из одной модели, листинг кода которой представлен в Приложении 8.

На рисунке 25 представлена модель Pothole, которая описывает единственную имеющуюся таблицу, хранящуюся объектно-реляционной базе данных PostgreSQL, используемую для хранения информации об обнаруженных ямах на дорогах.

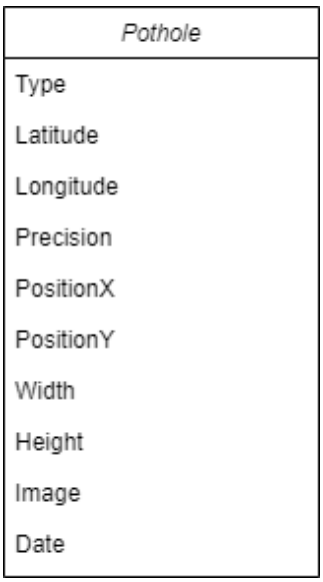


Рисунок 25. Модель Pothole.

Каждая обнаруженная яма сохраняется в базе данных со следующей информацией: тип неровности (на данный момент существует только один тип – яма), географические координаты автомобиля во время обнаружения неровности, точность обнаруженной неровности и ее позиция на изображении, дата обнаружения и наименования изображения.

В случае загрузки набора неровностей, находящихся на одном и том же изображении, создаются несколько записей в таблице, при этом дублирование изображений не осуществляется.

4.2. АЛГОРИТМ ДЛЯ ОПРЕДЕЛЕНИЯ ШИРИНЫ ЯМ

В ходе изучения способов измерения характерных размеров ям на изображениях, в условиях, где мы не знаем, на каком расстоянии от нас находится необходимый объект, было выявлено, что единственным решением данной проблемы является измерение размеров объекта относительно другого какого-либо объекта на изображении, размеры которого нам известны.

На основании того, что все снимки обучающей выборки были сделаны одной и той же камерой, под одним углом ее наклона на дорожном покрытии в США путем расчетов были получены значения количества сантиметров на пиксель изображения, на определенном горизонтальном участке дороги.

Размер исходных изображений составляет 3680 на 2760 пикселей. С целью уменьшения искажения размеров ям на дорожном покрытии, измерение размеров ширины обнаруженных ям будет выполняться в области, ограниченной следующим координатами: $x_1 = 900px$, $y_1 = 1600px$, $x_2 = 3100px$, $y_2 = 2000px$ (см. рисунок 26).

Рисунок 26. Ограничивающая область для вычисления ширины ям.

Ширина дорожной полосы в США равняется 3,67 метра. В обучающей выборке были подобраны изображения, дорожная полоса на которых близится к параллельности с курсом движения автомобиля.

По результатам работы модели нам известны размеры обнаруженных ям в пикселях. Так яма, обнаруженная на изображении 26, имеет следующие размеры – 270 на 175 пикселей.

Для вычисления количества сантиметров на пиксель необходимо: определить ширину дорожной полосы, в данном случае средняя ширина дорожной полосы в ограничивающей области приблизительно равна 1335 пикселей.

Путем деления ширины дорожной полосы на среднюю ширину дорожной полосы в пикселях, мы получим количество сантиметров на пиксель: $\frac{367}{1335} = 0{,}27$ см./пиксель.

Чтобы получить ширину ямы в пикселях необходимо перемножить количество пикселей на полученное значение. Именно для того, чтобы избежать большой погрешности была применена ограничивающая область, поскольку чем дальше яма находится от автомобиля, тем меньше кажутся ее размеры.

Таким образом, ширина обнаруженной ямы, представленной на рисунке 26, равна 74,2 см. При вычислении длины ямы аналогичным способом мы получим большую погрешность, потому что расчеты ведутся относительно горизонтального объекта, размеры которого нам были предварительно известны.

Данный алгоритм будет выдавать большую погрешность при использовании результатов текущих расчетов и изменении технических характеристик камеры, ее наклона и высоты расположения в автомобиле. При условии соблюдения текущих параметров расположения вычисленная ширина обнаруженной ямы будет приблизительно равна реальной.

4.3. РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ОБНАРУЖЕНИЯ ЯМ

На языке Python было написано приложение, при помощи которого выполняется обнаружение ям на входящем видеопотоке данных. Приложение выполняется на специальном устройстве – микрокомпьютере Raspberry Pi, к которому подключена видеокамера и GPS-трекер.

Модель нейронной сети, полученная в результате проведения эксперимента, описанного в главе 3, была конвертирована в фреймворке PyTorch в открытый формат представления нейронных сетей – ONNX при помощи функции torch.onnx.export.

При обнаружении ямы на дорожном покрытии срабатывает модуль, написанный на языке Python, который получает текущее местоположение

автомобиля с GPS-трекера. Листинг кода данного модуля представлен в Приложении 9.

Взаимодействие GPS-трекера с устройством осуществляется через последовательный порт. Данные между GPS приемниками и навигационными программами передаются по протоколу NMEA. В случае, когда полученные данные не содержат информации о текущем местоположении трекера, то модуль выдает соответствующую ошибку. Отправка данных на сервер в этом случае не осуществляется.

При обнаружении неровности на видеопотоке данных приложение сохраняет кадр видеозаписи, на котором была обнаружена яма. Загрузка данных в хранилище выполняется через API-интерфейс серверной части приложения, описанного в пункте 4.1.2, путем отправки POST-запроса с информацией об обнаруженных неровностях.

ЗАКЛЮЧЕНИЕ

В рамках выполнения выпускной квалификационной работы была изучена предметная область, а именно:

- проведен анализ существующих технических и технологических решений, способных выполнять обнаружение ям на дорожном покрытии;
- изучен государственный стандарт, устанавливающий требования к состоянию дорожного покрытия.

Для обучения моделей нейронных сетей был подготовлен набор данных, состоящий из снимков с ямами на дорожном покрытии. На этапе разметки данных был изучен инструмент Roboflow, позволяющий выполнять графическую аннотацию данных, их предобработку и аугментацию. Был подготовлен для обучения набор данных, состоящий из 1093 изображений.

Для проведения вычислительного эксперимента по изучению влияния архитектур нейронных сетей на точность обнаружения ям на дорогах было проведено исследование семейства сетей YOLO. На этапе обучения моделей нейронных сетей был изучен инструмент WandB, позволяющий отслеживать все системные показатели и показатели обучения в режиме реального времени.

По результатам проведения эксперимента была получена модель нейронной сети, удовлетворяющая заданным критериям. Полученная модель YOLOv5l показала точность равную 0.8429.

Для применения полученной модели было разработано приложение для обнаружения ям на дорожном покрытии, которые способно обрабатывать видеопоток данных с камеры, подключенной к интерфейсу USB и отправлять информацию об обнаруженных неровностях на удаленный сервер с фиксацией текущего местоположения, полученного от GPS-модуля.

Для дальнейшего мониторинга и оценки состояния дорожного покрытия был разработан сервис, позволяющий просматривать обнаруженные ямы на карте. Серверная часть приложения позволяет выполнять удаленную загрузку обнаруженных ям в хранилище данных.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Федеральная служба государственной статистики: [сайт]. URL: https://rosstat.gov.ru/ (дата обращения: 10.06.2022).
2. Российское информационное агентство: [сайт]. URL: https://tass.ru/obschestvo/7950737 (дата обращения: 10.06.2022).
3. Сведения о показателях состояния безопасности дорожного движения: [сайт]. URL: http://stat.gibdd.ru/ (дата обращения: 10.06.2022).
4. Шакирьянов Э. Д. Компьютерное зрение на Python. Первые шаги. Москва: Лаборатория знаний, 2021. 163 с.
5. Сергеев В. В. Обнаружение объектов на изображении: методические указания. Самара: СГАУ, 2010. 23 с.
6. Николенко С., Кадурин А., Архангельская Е. Глубокое обучение. Санкт-Петербург: Питер, 2018. 480 с.
7. McCulloch S.W., Pitts H.W. Logical calculus of the ideas immanent in nervous activity. // Bull. Math. Biophys. 1943. Vol. 5. P. 115-133.
8. Хайкин С. Нейронные сети: полный курс, 2е издание: Пер. с англ. Москва: Издательский дом "Вильямс", 2006. 1104 с.
9. Churchland P.S., Sejnowski T.J. The Computational Brain. Cambridge: MIT Press, 1992. 558 p.
10. M. Lewis. Man and Machine Vision. New York: McGraw-Hill, 1985. 574 p.
11. D. Marr. Vision. New York: MIT Press, 2012. 428 p.
12. N. Suga. Cortical computational maps for auditory imaging. // Neural nerworks. 1990. Vol. 3. P. 3-21.
13. N. Suga. Computations of velocity and range in the bat auditory system for echo location. // Computational Neuroscience. 1990. P. 213-231.
14. Головко В.А. Нейронные сети: обучение, организация и применение. Москва: Радиотехника, 2001. 188 с.
15. Реализация искусственных нейронных сетей в Линукс: [сайт]. URL: https://novainfo.ru/article/2010 (дата обращения: 10.06.2022).

16. ГОСТ Р 50597–2017. Дороги автомобильные и улицы. Требования к эксплуатационному состоянию, допустимому по условиям обеспечения безопасности дорожного движения. Методы контроля. Москва: Стандартинформ, 2017. 28 с.
17. Людовских Д.С. Система мониторинга качества дорожного покрытия с помощью мобильных устройств. Ростов-на-Дону: ЮФУ, 2016. 45 с.
18. Lanjewar B., Khedkar J., Sagar R., Pawar R., Gosavi K. Survey of Road Bump and Intensity Detection algorithms using Smartphone Sensors. // International Journal of Computer Science and Information Technologies. 2015. Vol. 6. P. 5133-5136.
19. Hoffmann M., Mock M., May M. Road-quality classification and bump detection with bycicle-mounted smartphones. // Proceedings of the 3rd International Conference on Ubiquitous Data Mining. 2013. Vol. 1088. P. 39-43.
20. Eriksson J., Girod L., Hull B., Newton R., Madden S., Balakrishnan H. The Pothole Patrol: Using a Mobile Sensor Network for Road Surface Monitoring. // MobiSys '08: Proceedings of the 6th international conference on Mobile systems, applications, and services. 2008. P. 29-39.
21. Prasad V., Kumari S. Pothole Detection using LiDAR. // Adv Automob Eng. 2021. Vol. 10. P. 10-13.
22. Kang B., Choi S. Pothole detection system using 2D LiDAR and camera. // International Journal of Innovative Science and Research Technology. 2017. Vol. 6. P. 1385-1393.
23. Ravi R., Habib A., Bullock D. Pothole mapping and patching quantity estimates using lidar-based mobile mapping systems. // Environmental Science. 2020. Vol. 2674. P. 124-134.
24. Chang K., Chang J., Liu J. Detection of pavement distresses using 3D laser scanning technology. // Trans-portation Research Record. 2020. Vol. 2674. P. 124–134.

25. Parker J. R. Algorithms for Image Processing and Computer Vision. India: Wiley Pub. 2011. 465 p.
26. Paragios N., Tziritas G. Adaptive detection and localization of moving objects in image sequences. // Signal Processing: Image Communication. 1999. Vol 14. P. 277-296.
27. Ansari S. Building A Realtime Pothole Detection System Using Machine Learning and Computer Vision. 2021. URL: https://towardsdatascience.com/building-a-realtime-pothole-detection-system-using-machine-learning-and-computer-vision-2e5fb2e5e746 (дата обращения 10.06.2022)
28. Aparna A., Yukti B., Rachna R., Varun G., Naveen A. Convolutional Neural Networks Based Potholes Detection Using Thermal Imaging. // Journal of King Saud University. Computer and Information Sciences. 2019. Vol. 34. P. 578-588.
29. Fan R., Wang H., Wang Y., Liu M., Pitas I. Graph Attention Layer Evolves Semantic Segmentation for Road Pothole Detection: A Benchmark and Algorithms. Computer Vision and Pattern Recognition. // IEEE Transactions on Image Processing. 2021. Vol. 30. P. 8144-8154.
30. Rosenblatt F. The perceptron: a probabilistic model for information storage and organization in the brain. 1958. Vol. 65. P. 386-408.
31. Hubel D. H., Wiesel T.N. Brain mechanisms of vision. // Scientific American. 1979. Vol. 241. P. 150-163.
32. Вакуленко С. А., Жихарева А. А. Практический курс по нейронным сетям. Санкт-Петербург: Университет ИТМО, 2018. 71 с.
33. Vehicle Type Recognition from image data: [сайт]. URL: https://github.com/ChristianJavierMelo/Vehicle-Type-Recognition (дата обращения: 10.06.2022).

34. Redmon J., Divvala S., Girshick R. You Only Look Once: Unified, Real-Time Object Detection. // Computer Vision and Pattern Recognition. 2016. P. 779-788.
35. Репозиторий Scaled-YOLOv4 // GitHub: [сайт]. URL: https://github.com/WongKinYiu/ScaledYOLOv4 (дата обращения: 10.06.2022).
36. Репозиторий YOLOv5 // GitHub: [сайт]. URL: https://github.com/ultralytics/yolov5 (дата обращения: 10.06.2022).
37. Wang C. Scaled-YOLOv4: Scaling Cross Stage Partial Network. // 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2021. P. 13024-13033.
38. D. Cochard. YOLOv5: The Latest Model for Object Detection: [сайт]. URL: https://medium.com/axinc-ai/yolov5-the-latest-model-for-object-detection-b13320ec516b (дата обращения: 10.06.2022).
39. Talukder D., Jahara F. Real-Time Bangla Sign Language Detection with Sentence and Speech Generation. // 2020 23rd International Conference on Computer and Information Technology (ICCIT). 2020. P. 1-6.
40. Nienaber S., Booysen M., Kroon R., Detecting potholes using simple image processing techniques and real-world footage. // Cognitive Informatics and Soft Computing. 2015. P. 893-902.
41. Nienaber S., Kroon R., Booysen M., A Comparison of Low-Cost Monocular Vision Techniques for Pothole Distance Estimation. // 2015 IEEE Symposium Series on Computational Intelligence. 2015. P. 419-426.
42. Documentation Roboflow: [сайт]. URL: https:// docs.roboflow.com (дата обращения: 10.06.2022).
43. Weiss K., Khoshgoftaar T., Wang D. A survey of transfer learning. Journal of Big Data. 2016.

44. Weights & Biases - Documentation // WandB: [сайт]. URL: https://docs.wandb.ai/ (дата обращения: 10.06.2022).

45. Getting Started – React // React – A JavaScript library for building user interfaces: [сайт]. URL: https://reactjs.org/docs/getting-started.html (дата обращения: 10.06.2022).

46. Documentation Mapbox // Maps, geocoding, and navigation APIs & SDKs Mapbox: [сайт]. URL: https://docs.mapbox.com/ (дата обращения: 10.06.2022).

47. Хук состояния // Изучение React. Полное руководство по React: [сайт]. URL: https://learn-reactjs.ru/core/hooks/state-hook (дата обращения: 10.06.2022).

48. Хук эффекта // Изучение React. Полное руководство по React: [сайт]. URL: https://learn-reactjs.ru/core/hooks/effect-hook (дата обращения: 10.06.2022).

49. Documentation Node JS // Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine: [сайт]. URL: https://nodejs.org/en/docs/ (дата обращения: 10.06.2022).

Приложение 1

Категории дорог и улиц городов и сельских поселений

Группы улиц	Категории дорог и улиц городов и сельских поселений
А	Магистральные дороги скоростного движения, магистральные улицы общегородского значения непрерывного движения
Б	Магистральные дороги и магистральные улицы общегородского значения регулируемого движения
В	Магистральные улицы районного значения транспортно-пешеходные
Г	Магистральные улицы районного значения пешеходно-транспортные, поселковые дороги
Д	Улицы и дороги местного значения (кроме парковых), главные улицы, улицы в жилой застройке основные
Е	Улицы в жилой застройке второстепенные, проезды основные, велосипедные дорожки

Приложение 2

Размеры дефектов покрытия и сроки их устранения

Вид дефекта	Группа улиц	Размер	Срок устранения, сут, не более
Отдельное повреждение (выбоина, просадка, пролом) длиной 15 см и более, глубиной 5 см и более, площадью, м^2, равной или более	А	0,06	1
	Б		3
	В		5
	Г		7
	Д		10
	Е		12
Повреждения (выбоины, просадки, проломы) площадью менее 0,06 м2, длиной менее 15 см, глубиной менее 5 см на участке полосы движения длиной 100 м, площадью, м2, более	А, Б	0,1	5
	В	0,5	7
	Г	0,8	10
	Д	2,1	14
	Е	5,2	20
Отклонение по вертикали крышки люка относительно поверхности проезжей части, см, более	Все	1,0	1
Отклонение по вертикали решетки дождеприемника относительно поверхности лотка, см, более			2

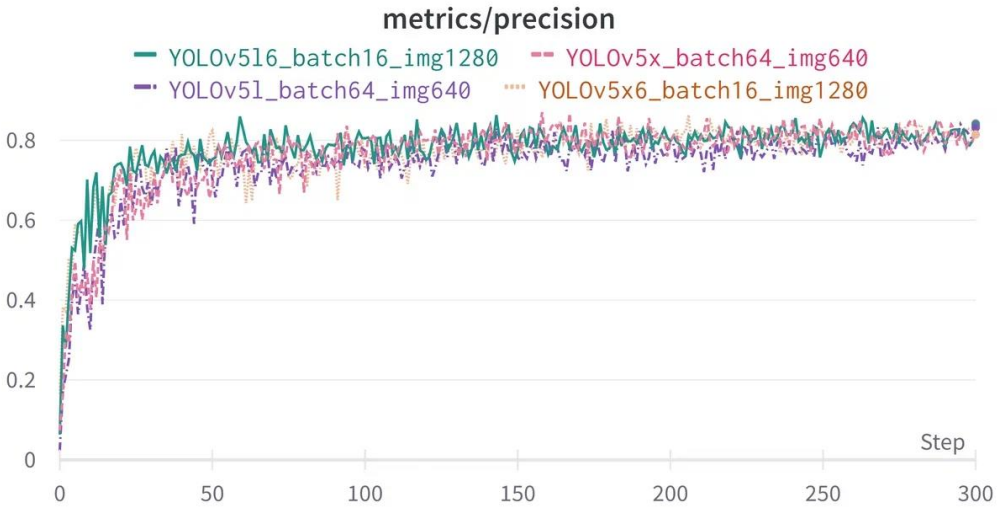
Приложение 3

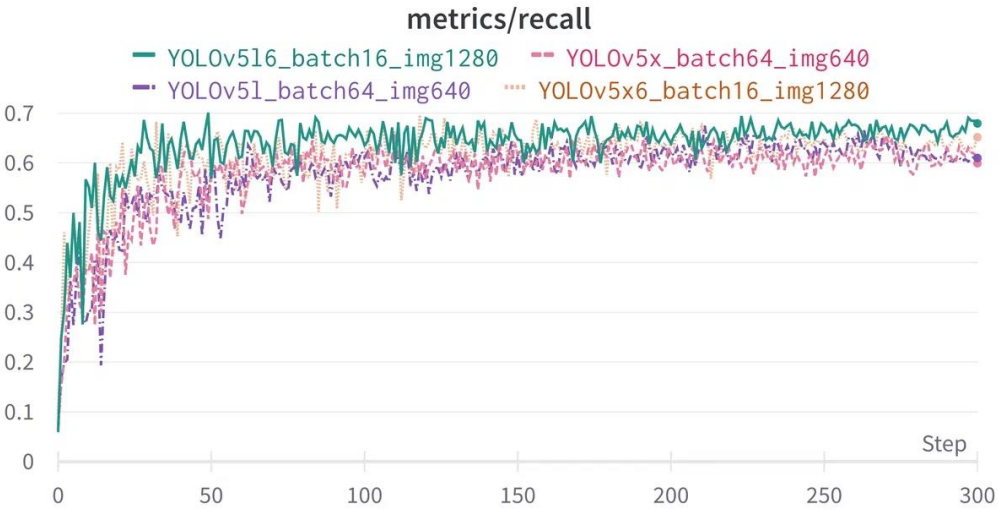
Технические характеристики сервера

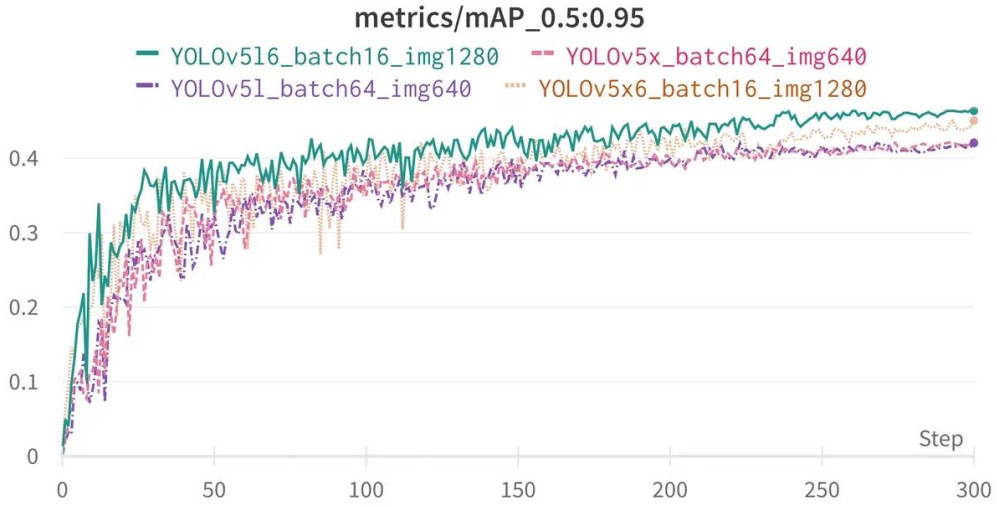
Наименование	Значение
Видеокарта	NVIDIA Tesla V100
Объем видеопамяти	32768 МБ
Ядер Tensor	43 тысячи
Производительность операции двойной точности	7 терафлопс в секунду
Производительность операции одинарной точности	14 терафлопс в секунду
Производительность глубокого обучения	112 терафлопс в секунду

Приложение 4

Результаты обучения моделей на обнаружение ям

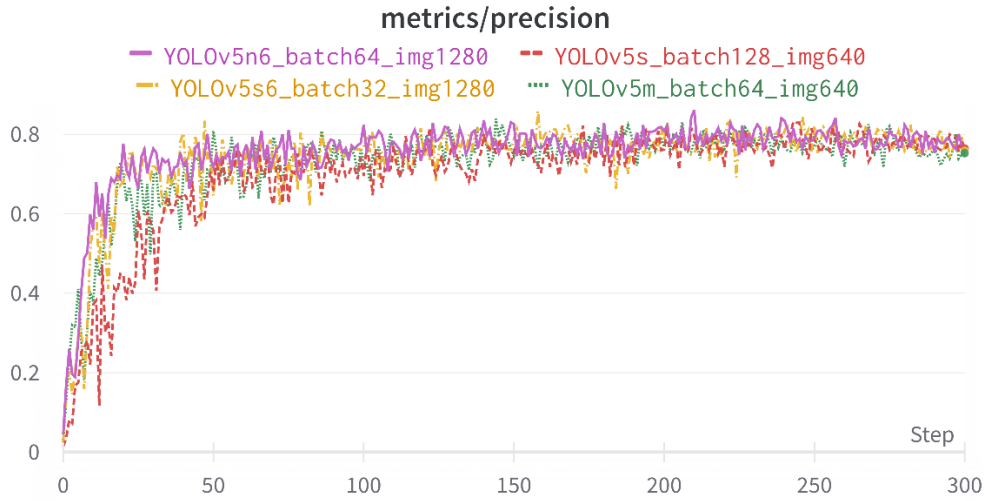


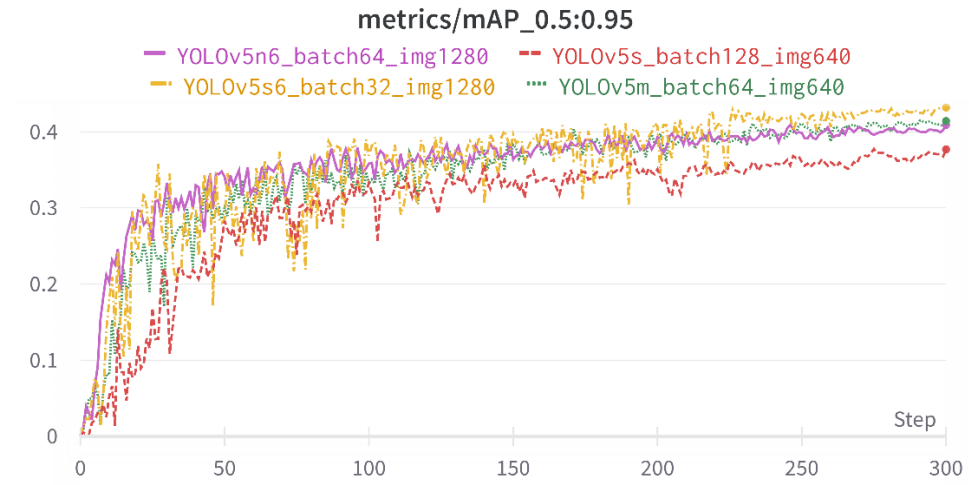


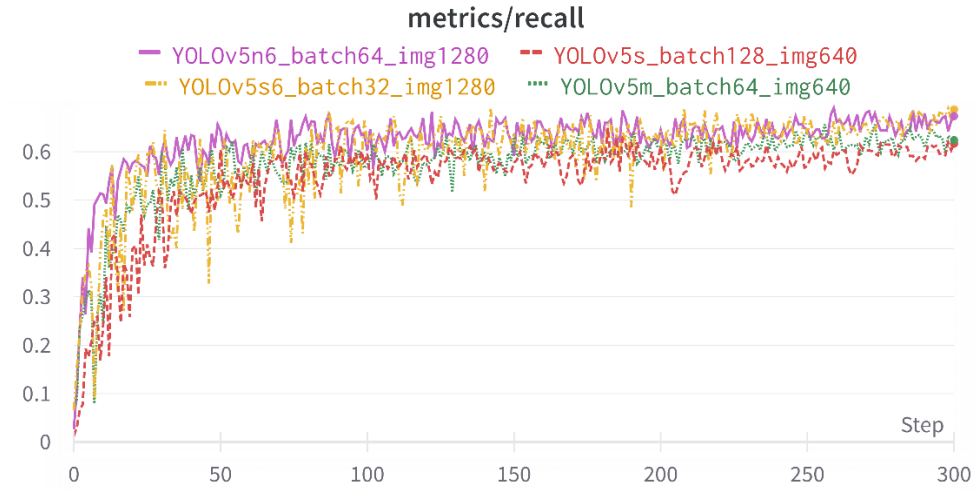


Приложение 5

Результаты обучения моделей на обнаружение ям

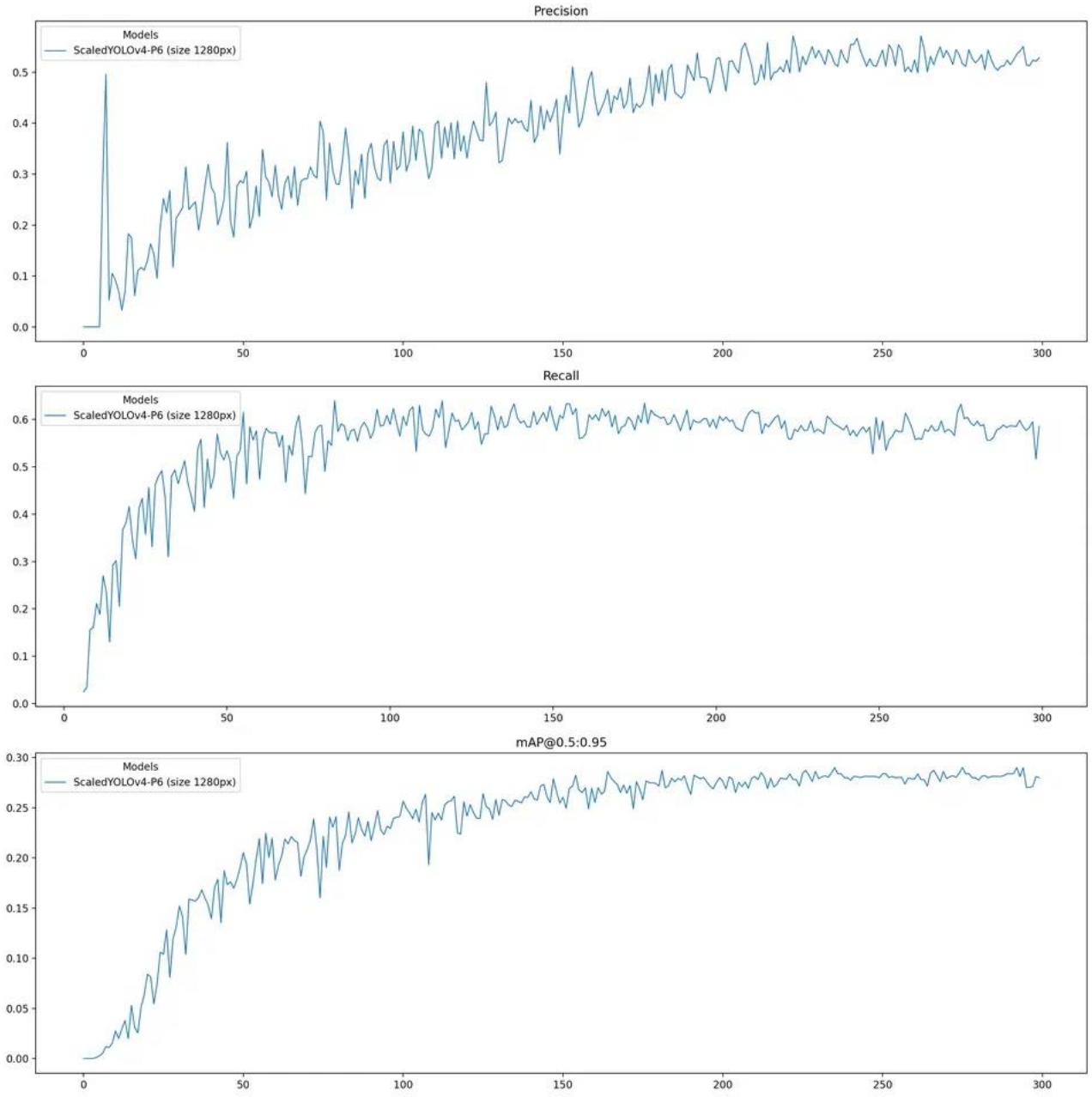






Приложение 6

Результаты обучения модели Scaled-YOLOv4 на обнаружение ям



Приложение 7

Листинг кода основного компонента Map

```
import React, { useRef, useEffect, useState } from 'react';
import mapboxgl from 'mapbox-gl';
import './Map.css';
mapboxgl.accessToken = 'pk.eyJ1IN2pmbDZmangifQ.-g_vE53SD2WrJmA';

function timeConverter(timestamp) {
    const a = new Date(timestamp * 1000);
    const months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug',
'Sep', 'Oct', 'Nov', 'Dec'];
    const year = a.getFullYear();
    const month = months[a.getMonth()];
    const date = a.getDate();
    const hour = a.getHours();
    const min = a.getMinutes();
    const sec = a.getSeconds();
    const time = date + ' ' + month + ' ' + year + ' ' + hour + ':' + min
+ ':' + sec;
    return time;
}
const Map = () => {
    const [potholes, setData] = useState({ hits: [] });
    useEffect(async () => {
        const result = await axios(
            'http://localhost/api/potholes',
        );
        setData(result);
    });
    const mapContainerRef = useRef(null);
    const [lng, setLng] = useState(65.5655);
    const [lat, setLat] = useState(57.1510);
    const [zoom, setZoom] = useState(13);
    useEffect(() => {
        const map = new mapboxgl.Map({
            container: mapContainerRef.current,
            style: 'mapbox://styles/mapbox/light-v10',
            center: [lng, lat],
            zoom: zoom
        });
     ...
        return () => map.remove();
    }, []);
    return (
        <div>
            <div className='sidebarStyle'>
                 Longitude: {lng} | Latitude: {lat} | Zoom: {zoom}
            </div>
            <div className='map-container' ref={mapContainerRef} />
        </div>
    );
};
export default Map;
```

Приложение 8

Листинг кода модели Pothole

```
const dbConfig = require('../config/db.config');
const Sequelize = require('sequelize');
const sequelize = new Sequelize(dbConfig.DB, dbConfig.USER, dbConfig.PASSWORD, {
  host: dbConfig.HOST,
  port: dbConfig.PORT,
  dialect: dbConfig.dialect,
  operatorsAliases: false,
  pool: {
    max: dbConfig.pool.max,
    min: dbConfig.pool.min,
    acquire: dbConfig.pool.acquire,
    idle: dbConfig.pool.idle
  }
});
const Pothole = sequelize.define('collection', {
        name: {
            type: Sequelize.STRING
        },
        precision: {
            type: Sequelize.DOUBLE
        },
        image: {
            type: Sequelize.STRING
        },
        latitude: {
            type: Sequelize.DOUBLE
        },
        longitude: {
            type: Sequelize.DOUBLE
        }
    });
const db = {};
db.Sequelize = Sequelize;
db.sequelize = sequelize;
db.potholes = Pothole;

module.exports = db;
```

Приложение 9

Листинг кода модуля получения текущего местоположения автомобиля

```
import serial

SERIAL = "/dev/serial0"

def getCoordinates():
    gps = serial.Serial(SERIAL, baudrate = 115200, timeout = 1)
    result = None
    try:
        line = gps.readline()
        typeOfMessage = line[:6]
        if typeOfMessage == '$GPRMC':
            values = line.split(',')
            latitude = values[3]
            longitude = values[5]
            result =  { "result" : True, "data": {"longitude": longitude,
"latitude": latitude} }
        else:
            result = { "result" : False }
    except:
        result = { "result" : False }
    finally:
        gps.close()
        return result
```