Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Факультет математики, механики и компьютерных технологий
Кафедра прикладной математики и программирования
Направление подготовки: 01.04.02 Прикладная математика и информатика

РАБОТА ПРОВЕРЕНА
Рецензент, доцент кафедры АТ,
к.т.н., доцент
_____/В.Д. Шепелёв
«____»_____20___г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д.ф.-м.н.,
профессор
_____/А.А.Замышляева
«____»_____20___ г.

Контроль сверхнормативных выбросов и перемещения транспортных средств на основе данных, полученных нейронной сетью в видеопотоке реального времени

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ–01.04.02.2020.077.ПЗ ВКР

Руководитель работы, к.ф.-м.н.,
доцент
_____/Т.В. Карпета
«____»_____2020 г.

Автор работы
Студент группы ЕТ-222
_____/И.Г. Чарбадзе
«____»_____2020 г.
Нормоконтролер,
ст. преподаватель
_____/Н.С. Мидоночева
«____»_____2020 г.

Челябинск
2020

АННОТАЦИЯ

Чарбадзе И.Г. Контроль сверхнормативных выбросов и перемещения транспортных средств на основе данных, полученных нейронной сетью в видеопотоке реального времени. – Челябинск: ЮУргУ, ЕТ-222, 57 с., 42 ил., библиогр. список – 22 наим.

Целью данной работы является разработка алгоритмов подсчета транспортных средств, скорости и экологических выбросов. Решается задача автоматического сбора данных о ситуации на перекрестке в режиме реального времени.

В первом главе был проведен обзор основных способов подсчета транспортных средств их скоростей и экологических выбросов. Рассмотрены существующие системы подсчета ТС, скоростей и их выбросов. Проанализированы их плюсы и минусы рассмотренных решении. Изучив полученные результаты выбраны нейросетевой подход для решения поставленной цели.

Во второй главе были разработаны алгоритмы подсчета ТС, скорости и выбросов. Рассмотрена работа нейронной сети, которая выдает данные для работы разработанных алгоритмов.

В третьей главе был подсчитана точность алгоритмов подсчета ТС и расчета расстояния и приведены результаты расчета выбросов.

ОГЛАВЛЕНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ТС – транспортное средство;

ПО – программное обеспечение;

НС – нейронная сеть;

ИТС – интеллектуальная транспортная система;

ПК – персональный компьютер;

CNN – Convolution neural network;

SVM – Support vector machine;

PNR – Region proposal network;

SSD – Single shot detector.

ВВЕДЕНИЕ

В условиях постоянно увеличивающегося количества транспорта в городе встает проблема быстрого реагирования на изменяющуюся ситуацию на дорогах и экологическое состояние в городе. В последние годы эти проблемы обсуждаются наиболее часто. Для отслеживания загруженности дорог и экологического состояния воздуха в городах используют различные кратковременные исследования что не позволяет понять влияние выбросов от транспортных средств в долгосрочном периоде.

В условиях ограничения бюджета нет возможности проводить такие исследования часто. Одним из решении этих проблем является внедрение интеллектуальной транспортной системы. ИТС постоянно следит за состоянием дорожного узла вычисляя показатели загруженности дорог, среднюю скорость потока и экологические выбросы накапливая большие данные. На основе собранных данных можно провести анализ эффективности пропускной способности дорожных узлов. Менять значение времени горения зеленого света светофора для предотвращения заторов вследствие чего снизится количество выбросов от ТС в области дорожного узла. Данная работа посвящена проблеме обработки данных, полученных от нейронной сети в режиме реального времени и их последующего анализа.

Целью данной работы является разработать модули подсчета транспортных средств, экологических выбросов и скорости во всех направлениях дорожного узла в режиме реального времени. Для достижения поставленных целей необходимо решить следующие задачи:

1) разработать алгоритм подсчета транспортных средств по направлениям и категориям (например, легковой автомобиль, грузовик и т. д.);

2) разработать алгоритм вычисления скорости автомобиля по видеопотоку;

3) разработать алгоритм расчета выбросов от ТС.

1 ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИИ ДЛЯ ПОДСЧЕТА ТРАНСПОРТНЫХ СРЕДСТВ, ВЫЧИСЛЕНИЯ СКОРОСТИ И ВЫБРОСОВ В РЕЖИМЕ РЕАЛЬНОГО ВРЕМЕНИ

1.1 Системы на основе радара

В основном данные системы находятся у ГИБДД для определения превышения скорости ТС. Все радары делятся на два типа: стационарный и мобильный. Мобильные радары можно легко перемещать и устанавливается в любом месте рядом с дорожным узлом. Стационарные радары-камеры жестко устанавливаются над дорожным узлом и фиксируют превышение скорости и фотографируют государственный номер ТС. Расположение данного типа радаров не меняется. Обнаружение и измерение скорости ТС происходит за счет изменения длинны волны отраженного радиосигнала – эффект Доплера. По изменению частоты радар вычисляет разность скоростей между самим радаром и объектом, вычисляя скорость объекта (рисунок 1.1).

Рисунок 1.1 – Эффект Доплера для движущегося объекта

Эффект Доплера – это изменение частоты и волны принимаемая приемником (радаром), вследствие движения объекта. Частота волны опред еляется следующим образом:

$$\omega = \frac{2\pi(c - v)}{w_0} = w_0 \frac{1}{1 - \frac{v}{c}},$$

где w_0 – угловая частота, с которой источник испускает волны;

c – скорость распространения волн в среде;

v – скорость источника волн относительно среды (положительная, если источник приближается к приёмнику и отрицательная, если удаляется).

Плюсом данного метода является точность определения скорости ТС. К минусам стоимость оборудования и невозможность определения класса.

1.1.1 Радарный комплекс «Арена»

Данный комплекс разработан в городе Санкт-Петербург, компанией ЗАО «Ольвия». Радарный комплекс «Арена» автоматический измеряет скорость ТС параллельно фотографируя их на расстоянии до 90 м. Данные о превышении скорости передаются по радиоканалу в ближайший пост ГИБДД. Радар измеряет скорость как приближающегося, так и удаляющегося ТС. Диапазон измерения скорости у данного комплекса от 20 до 250 км/ч.

Рисунок 1.2 – Радарный комплекс «Арена»

Особенностью данного комплекса является мобильность. Радар расположен на треноге что позволяет его легко перемещать. Данные о нарушении содержат изображение ТС нарушителя, дату и время превышения скорости.

Данный комплекс не фиксирует скорость всего потока что не позволяет использовать его для поставленных целей.

1.1.2 Датчик интенсивности «Аркен»

Датчик интенсивности «Аркен» разработан компанией ИТЦ-М (инженерное технический центр, город Минск).

Датчик «Аркен» использует метод радара (рисунок 1.3) для обнаружения и измерения скорости проехавших ТС [1]. Датчик «Аркен» обнаруживает ТС после чего присваивает ближайшую ему полосу движения.

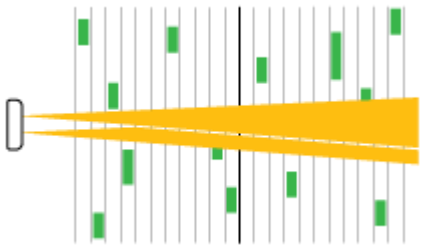

Рисунок 1.3 – Схема работы датчика

Для определения скорости датчик генерирует два луча, которые передаются параллельно друг другу, создавая «скоростную ловушку». Измерив время прохождения ТС через эти лучи с точностью до доли миллисекунды, которое используется для расчета скорости каждого отдельного ТС.

У двухлучевого датчика антенны расположены на фиксированном расстоянии равному 13,97 см. Это расстояние используется для расчета скорости. Излучая 2 параллельных луча датчик собирает отклики от объектов при пересечении 1-го и 2-го лучей (рисунок 1.3). В зависимости от направления движения один из лучей всегда будет пересечен немного раньше, чем второй (рисунок 1.4).

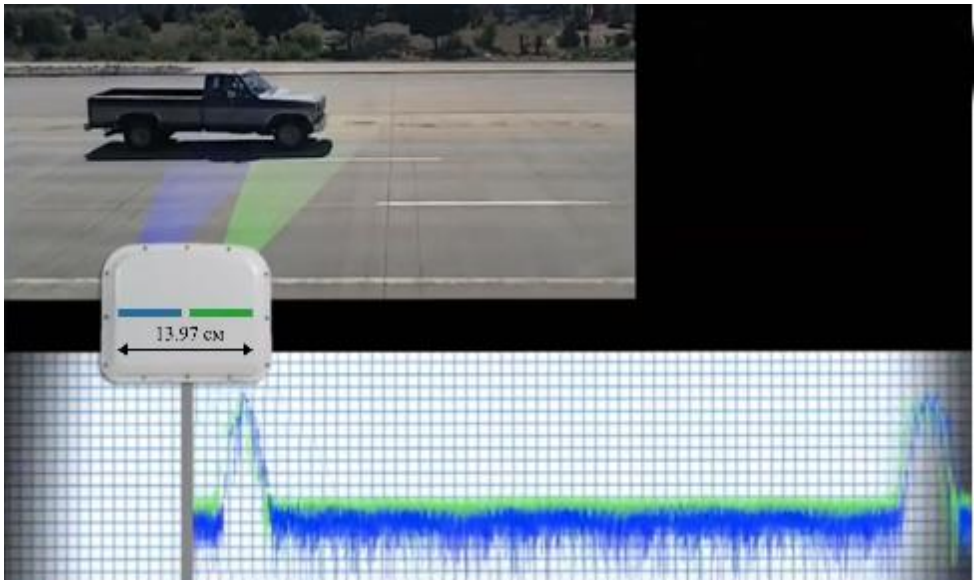

Рисунок 1.4 – Работа системы «Аркен»

Далее датчик производит синхронизацию сигналов и вычисляет время смешения. Это время используется для расчета скорости объекта (рисунок 1.5).

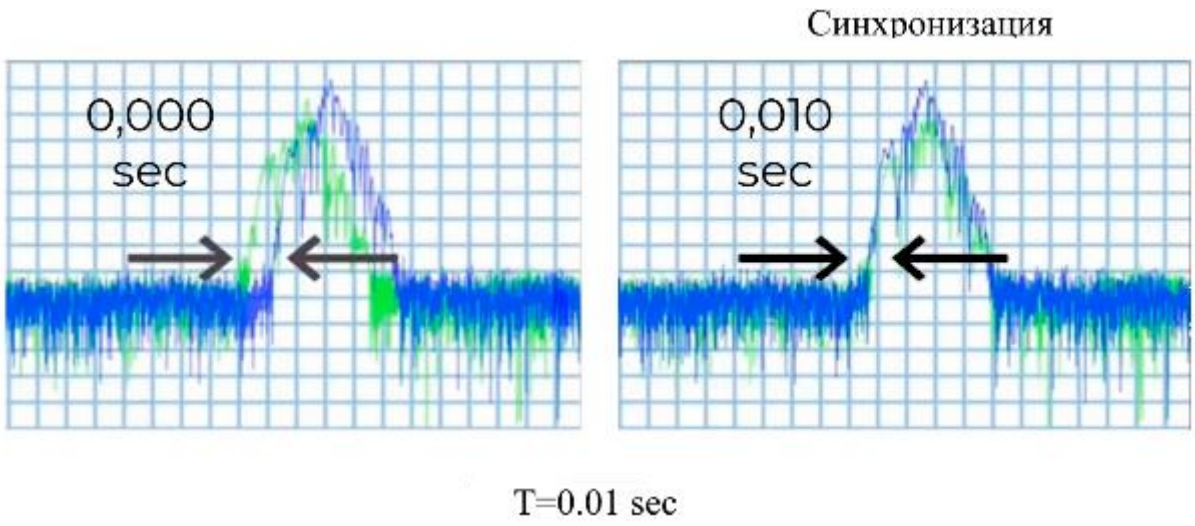

Рисунок 1.5 – Расчет смешения сигнала

Для определения скорости датчик не использует сложные алгоритмы и вычисления происходят мгновенно:

$$V = \frac{d}{T},$$

где d – расстояние между антеннами;

T – время смешения.

Анализируя время прохождения лучей датчик способен определять класс ТС по длине (до 8 типов) и направление движения (по очередности пересечения лучей).

Накопленные данные сохраняются с памяти датчика для последующего анализа и расчета различных параметров трафика, такие как количество ТС на промежуток времени, средняя скорость потока и другие.

У данной датчика есть несколько недостатков:

− невозможность определить дальнейшее движение транспорта;

− сложность использования в области перекрестка, так как на пути луча не должно быть объектов, мешающих нормальной работе детектора (кроны деревьев, дорожные знаки, рекламные щиты);

− датчик видит всего 2 направления движения ТС.

Преимущества:

− простота установки;

− точность определения скорости и классификации транспорта.

1.2 Программное обеспечение Avedex

Avedex разработан компанией Mallenom system [15]. Поставляется в виде программного обеспечения устанавливаемого на ПК пользователя. Она предназначена для автоматического сбора данных об интенсивности транспортного потока и его состава (рисунок 1.6). Система Avedex не использует физические датчики для подсчета транспортных средств и их классификации.

Основные функции:

− подсчет тс по видеоизображению;

− классификация автомобилей по 3 категории: легковые, грузовые, общественный транспорт;

− отображение статистики по трафику и экспорт статистики в форматах CSV и XML;

Перед обработкой видео файла необходимо расставить отрезки в места подсчета ТС.

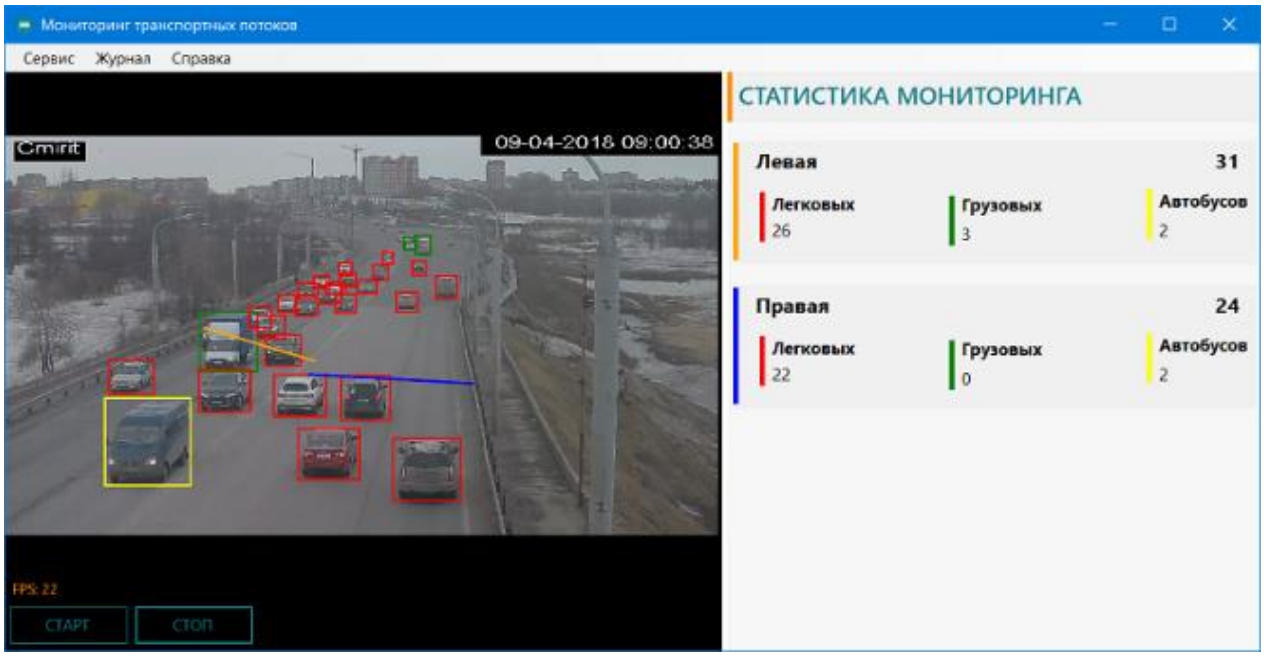

Рисунок 1.6 – Интерфейс программы Avedex

Возможности системы Avedex:

− подсчет производится полностью в автоматическом режиме;

− возможность подключения от 1 до 4 камер видеонаблюдения;

– подсчет можно вести как по видеопотоку онлайн, так и по видеофайлу;

– пользователь сам размечает дорожные полосы на изображении для подсчета трафика;

– экспорт статистики может быть произведен в краткой форме (суммарный трафик) или в расширенной форме (трафик с делением на категории).

Главный недостаток данного ПО то, что он определяет всего 3 вида транспорта (рисунок 1.6).

1.3 Сервис анализа дорожного трафика Traffic Data Air и Traffic Data Land

Программное обеспечение разработано компанией Traffic Data [15] и предназначено для получения информации о транспортном потоке из видеозаписей дорожных узлов. В данном ПО для обработки видео используются алгоритмы компьютерного зрения. ТС детектируются, классифицируются, строятся траектории их движения. На основании обработанного видео Traffic Data позволяет определить:

- количество ТС в потоке;
- состав потока;
- интенсивность потока.

Существует две вервии данной программы для стационарных камер и для видеозаписей, снятых с помощью квадрокоптера.

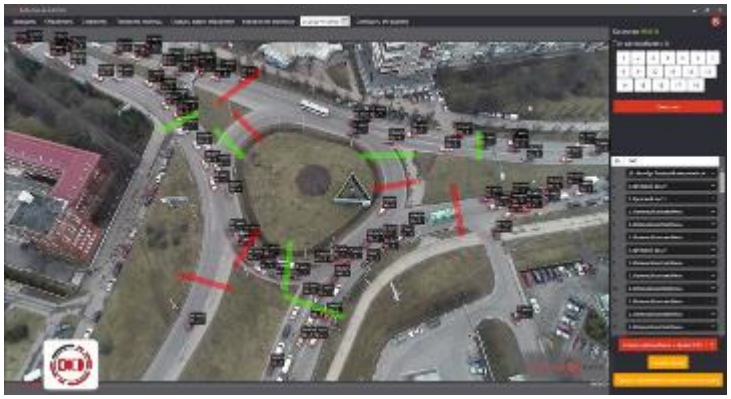

Рисунок 1.7 – Интерфейс программы Traffic Data Air

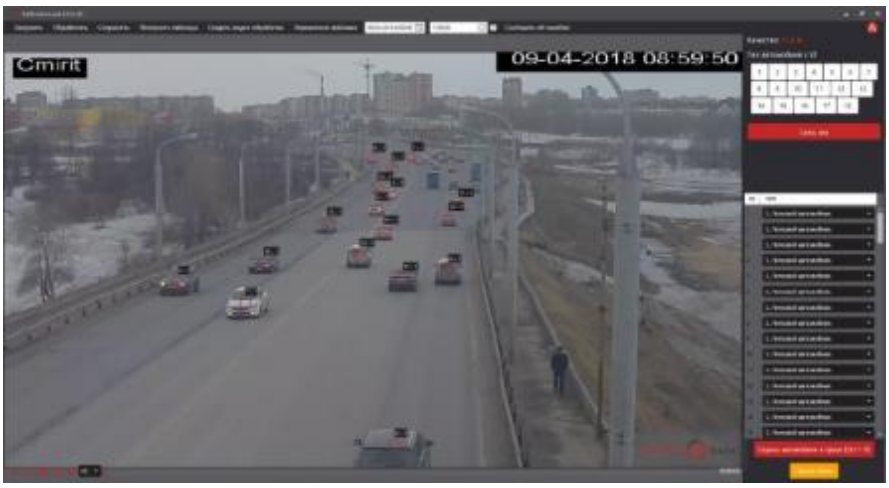

Рисунок 1.8 – Интерфейс программы Traffic Data Land

Для видеозаписей существуют требования. Для Traffic Data Air необходимо что бы камера была неподвижной чего трудно достичь, используя квадрокоптер. Максимальное отклонение от прямого угла в 45 градусов, но при этом качество распознавания падает

Для обработки видео системе нужно указать линии (створы) пересечения, в которых она будет считать ТС. Так же камера должна быть неподвижной для корректного просчета траектории движении ТС. Выгрузка данных происходит в формате Excel.

TrafficData Air и Land детектирует автомобили, размер которых больше пикселей. На рисунке 1.9 автомобили в зеленой зоне будут детектироваться, а в красной нет.

Рисунок 1.9 – Область определения ТС

Преимущества:

− точность определения. Из рисунка 6 видно, что точность определения ТС довольно высока;

− определение скорости;

− большое количество ТС, которое может определить Traffic Data до 18 типов.

Главным минусом данного ПО является то, что она может работать только с заранее записанными видеороликами.

1.4 Сервис определения скорости Car Crash Analysis

Сервис Car Crash Analysis запустился в тестовом режиме в 2019 году [16]. Данный сервис предназначен для анализа скорости ТС перед аварией (рисунок 1.10) на заранее записанном видеофайле. На данный момент система работает не в автоматическом режиме, каждый загруженный видеофайл необходимо подготовить перед расчетом скорости.

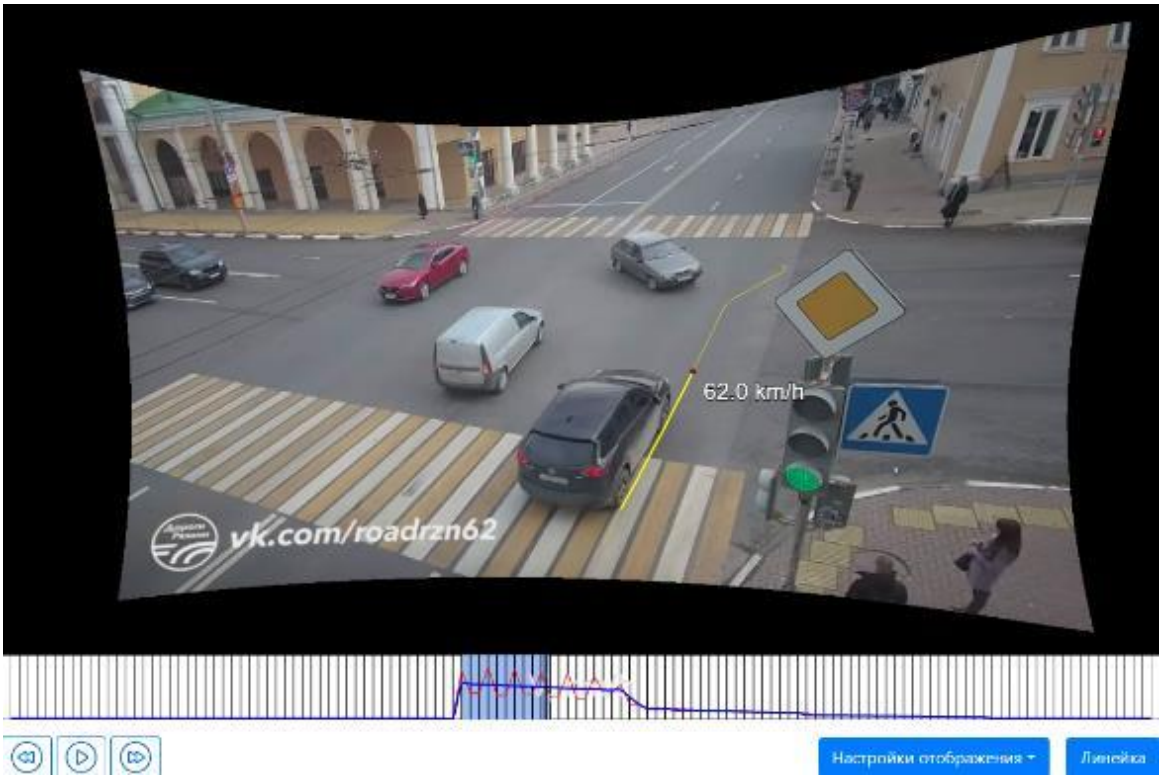

Рисунок 1.10 – Интерфейс программы CarCrashAnalysis

На первом этапе обработки видеофайла необходимо избавится от дисторсии. Дисторсия – это искажение прямых линии на изображении или видео. Данный этап происходит автоматически.

На втором этапе необходимо измерить расстояния на плоскости дороги. Для этого оператор находит расстояние ширины дороги, полосы движения используя ориентиры с видео и сравнивая их со спутниковыми картами.

На третьем этапе оператор на каждом кадре отмечает точками интересующий объект, отмечая траекторию движения, в итоге сервис рассчитывает текущую скорость и ее изменение во время движения.

Пока сервис работает только с видеозаписями, записанными с помощью стационарных камер.

Главные минусы данной системы:

− полуавтоматический режим работы. каждый загруженный видеоролик нужно размечать вручную, что заметно влияет на скорость работы;

− невозможность обрабатывать видео в реальном режиме;

− точность определения скорости сильно зависит от качества исходного видеоролика.

К плюсам можно отнести точность. Проведенные автором тесты показывают погрешность до 3%.

1.5 Мобильная станция анализа воздуха

В мобильной станции позволяет в реальном времени анализировать окружающий воздух. Мобильные станции можно объединить в сеть более чем 200 станции [17] (рисунок 1.11), что позволяет следить за перемещением загрязняющих веществ. Специальный блок в составе программного обеспечения позволяет стоить розу концентрации в точке расположения станции (рисунок 1.12).

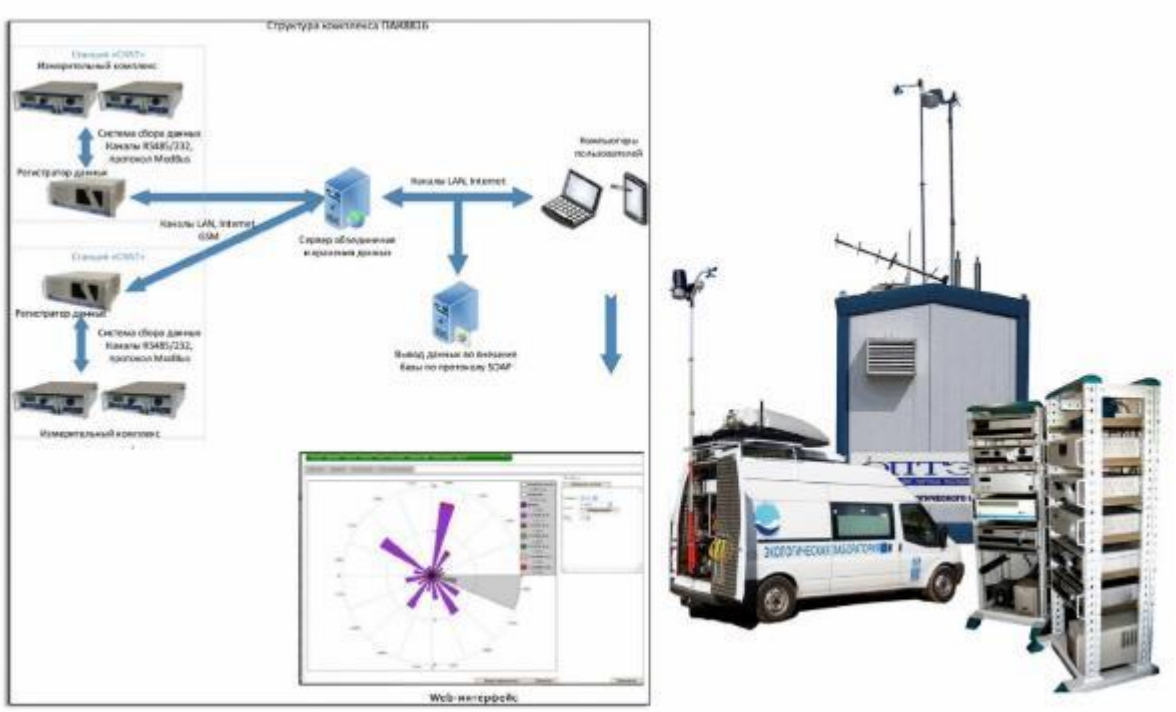

Рисунок 1.11 – Структура программно-аппаратного комплекса «СКАТ-РД»

Станция атмосферного мониторинга, как компонент системы, содержит метеокомплекс с датчиками давления и влажности воздуха, температуры, уровня осадков, направления и скорости ветра. Следует отметить, что для задач атмосферного мониторинга важнейшими каналами метеорологической информации являются направление и скорость ветра. Именно эти каналы позволяют в текущем режиме не только получать оперативную информацию на данный момент времени, но и смоделировать процессы адвекции примесей атмосферного воздуха в краткосрочном прогнозе, дать качественную перспективу возможных тенденций изменения климата.

Преимущества данного комплекса – это точность определения выбросов в точке установки стенда. Главным недостатком является стоимость оборудования и станции.

1.6 Разновидность газоанализаторов

В зависимости от физического принципа, по которому осуществляется анализ газовой среды для выявления отдельных компонентов, существует

больше 10 разновидностей газоанализаторов [18, 19]. Не существует полностью универсальной конструкции, которая бы позволяла анализировать состав любых смесей. Для одних разновидностей газов используется один физический принцип, в то время как для других он не действенный или небезопасный.

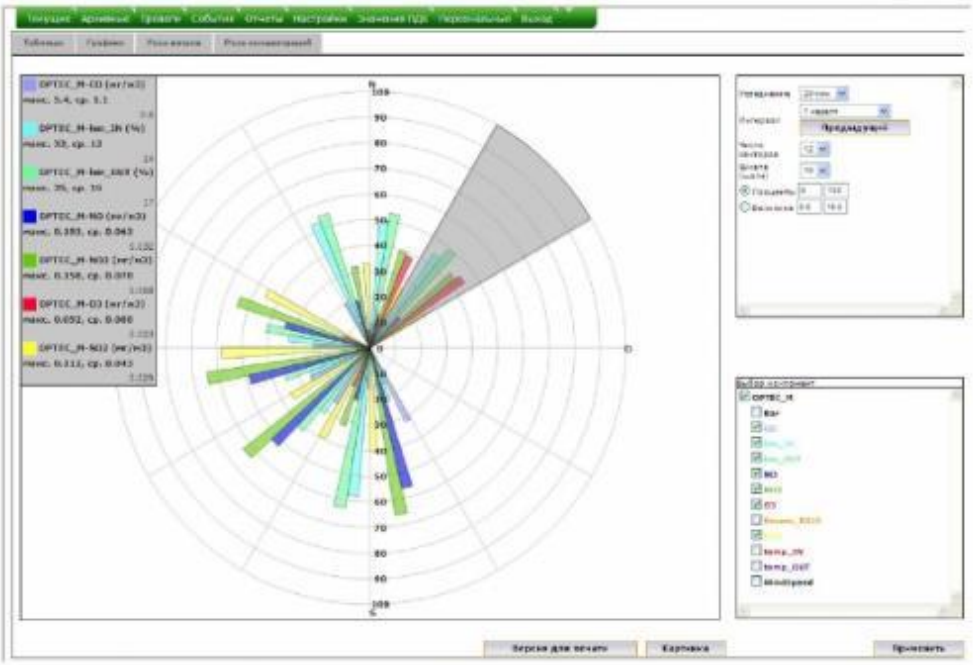

Рисунок 1.12 – Web-интерфейс пользователя, показывающий розы концентрации примесей в воздухе

Выделяют следующие виды газоанализаторов:

− магнитный газоанализатор используется для анализа кислорода. Подобные приборы используются в различных высокотехнологических механизмах, в которых проводится подготовка газовой смеси для сжигания. По данному принципу работает лямбда-зонд, который монтируется в выхлопной системе современных автомобилей. Устройство определяет концентрацию кислорода в выхлопных газах, что позволяет оценить, насколько эффективно прогорело топливо;

− инфракрасные облучают газовую среду инфракрасными лучами, после чего чувствительные датчики реагируют на уровень поглощения молекулами вещества излучаемого света. Такие устройства имеют взрывозащищенный корпус, поэтому часто используются с взрывоопасными

веществами. По данному принципу работает значительная доля лабораторного и промышленного оборудования;

− ультрафиолетовые работают по схожему принципу с инфракрасными, за тем исключением, что осуществляют облучение ультрафиолетовыми лучами. Данные приборы также анализируют интенсивность поглощения молекулами измеряемой среды направленных на них лучей;

− и другие.

Рисунок 1.13 – Пример ручного газоанализатора

Главным минусов данных газоанализаторов является стоимость одного прибора. А также так как для работы данных систем необходимо присутствие человека нет возможности замерять выбросы в атмосферу круглосуточно.

1.7 Нейросетевые подходы к решению задач обнаружения объектов

1.7.1 Нейронная сеть R-CNN

Архитектура R-CNN [10, 12, 13] была разработана командой из UC Berkley для применения CNN к задаче обнаружения объектов (рисунок 1.14). В качестве CNN-сети используется готовая архитектура – CaffeNet [2].

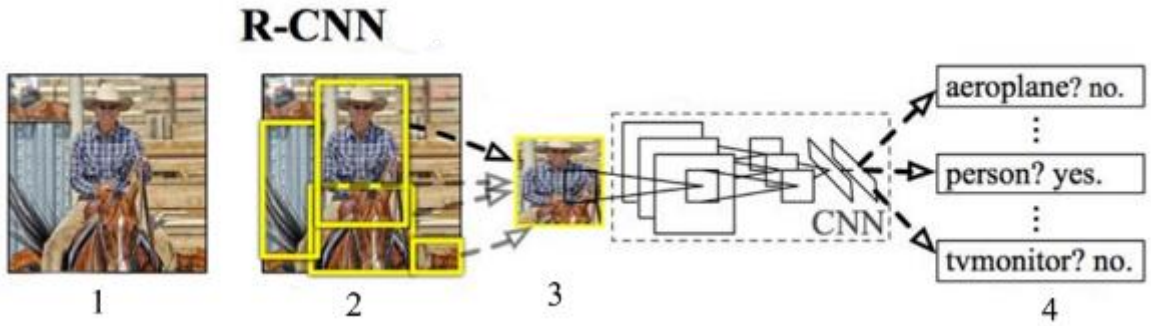

Рисунок 1.14 – Схема работы R-CNN 1. Входное изображение. 2. Предполагаемые регионы. 3. Вычисление CNN-признаков. 4. Классификация

Для решения задачи распознавания на вход сети подается не все изображение, а ее область в которых возможно содержится интересующийся нам объект. Выбор таких областей происходит с помощью алгоритма selective search [8]. Данный алгоритм генерирует 2000 различных областей, в которых с наибольшей вероятностью содержится объект. Так как CaffeNet на вход принимает изображения фиксированного размера 227x227 пикселей, поэтому входной изображение сжимается до квадрата (рисунок 1.15) и подаются на вход CNN, которая на выходе дает 4096 мерный вектора признаков. Далее вектор признаков подается на вход в SVM, обученных для каждого класса и выполняя классификацию.

Рисунок 1.15 – Преобразование региона в квадрат

Главным недостатком данной сети, которая не дает использовать ее под поставленные задачи – это время обработки одного кадра (около 49 секунд на изображение). Следующее, но не менее важный минус – время обучения и высокие требования к вычислительным ресурсам. Что бы решить эти проблемы было разработано две модификации сети – Fast и Faster R-CNN.

1.7.2 Нейронная сеть Fast R-CNN

Так как скорость обработки R-CNN оставляет желать лучшего, авторы разработали улучшенную модель Fast R-CNN (рисунок 1.16), которая отличается от R-CNN двумя модификациями:

− через CNN регионы пропускается не по отдельности, а все изображение целиком, что позволяет проводить операцию свертки один раз для каждого изображения;

− Теперь обучение происходит в процесс, вместо 3-х независимых моделей (CNN, SVM, bbox regressor).

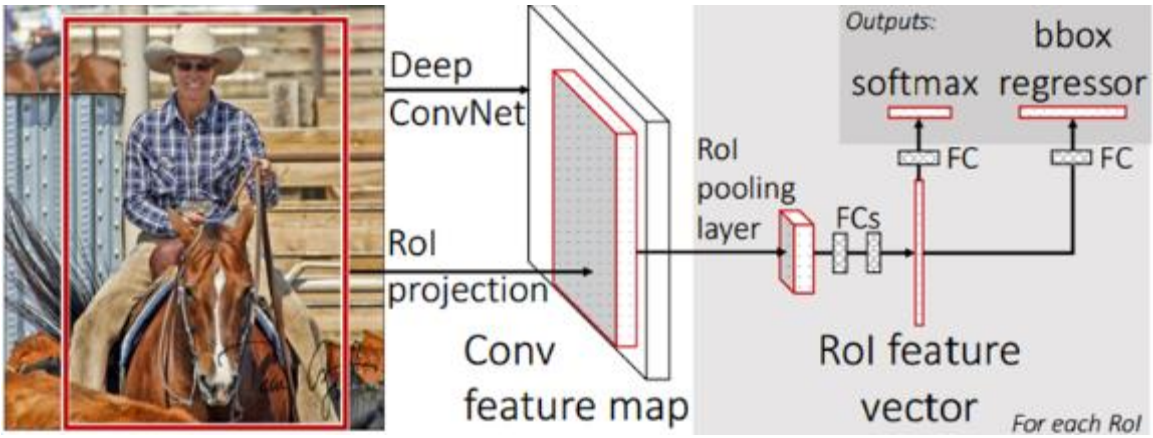

Рисунок 1.16 – Процесс работы Fast R-CNN

Внесенный модификации улучшили архитектуру и значительно улучшили результаты производительности, сократив время обработки изображения до 2.5 секунды на изображение.

Несмотря на внесенные изменения и улучшения скорости обработки изображения этого не хватает для применения данной архитектуры в режиме реально времени.

1.7.3 Нейронная сеть Faster R-CNN

Улучшения в Faster R-CNN еще большей скорости обработки изображения. Разработки предложили ввести специальную сеть для предложения регионов, называемой PNR и поместили ее после последнего сверточного слоя (рисунок 1.17). Эта сеть позволяет генерировать предлагаемые регионы на основе лишь последней сверточной карты

признаков. Предсказанные регионы далее изменяются с использованием слоя RoI, результат которого используется для классификации.

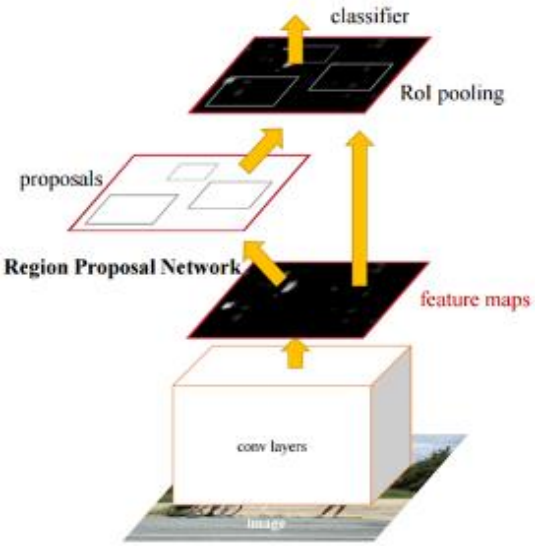

Рисунок 1.17 – Архитектура Faster R-CNN

Как видно из результатов тестирования (рисунка 1.18) скорость обработки изображения намного уменьшилось до 0.2 секунды на изображение, что при использовании данной сети в режиме реального времени даст сильное отставание со временим.

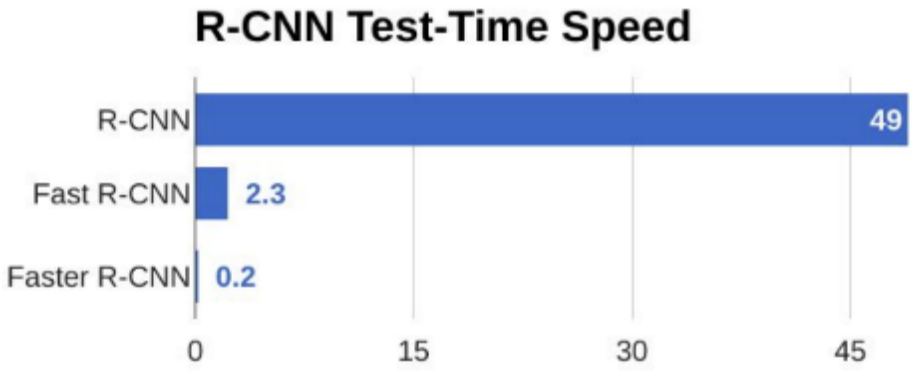

Рисунок 1.18 – Результаты производительности семейства архитектур R-CNN

1.8 Нейронная сеть YOLO v3

YOLO v3 имеет совсем другой принцип работы в отличии от описанных выше. Отличительной особенностью данной нейронной сети является то, что она применяет CNN один раз и ко всему изображению сразу вместо того, чтобы примерять CNN несколько раз к разным регионам

изображения. Сеть делить изображение на сетку и предсказывает bounding boxes и вероятность того, что там находится нужный нам объект для каждого bounding boxes.

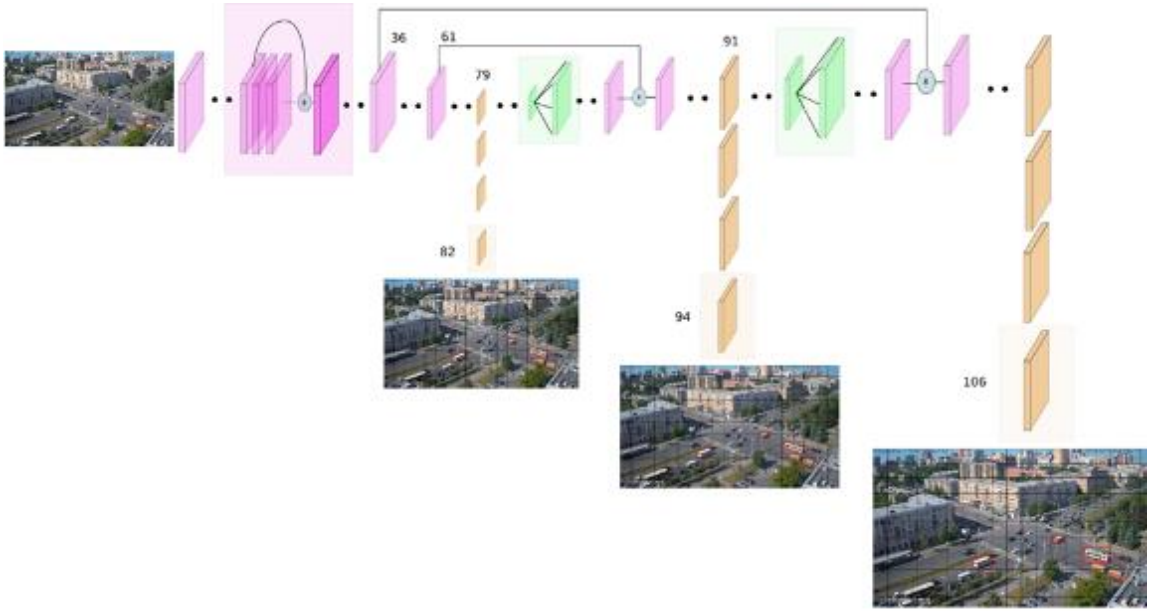

Рисунок 1.19 – Архитектура нейронной сети YOLO

Хотя YOLO немного проигрывает в точности классификации объектов, ее главным плюсом является скорость обработки изображения, которая дает возможность использовать ее в режиме реального времени.

1.9 Нейронная сеть SSD

Нейронная сеть SSD [21] использует наиболее удачные решения в архитектуре YOLO (например, non-maximum suppression, которая удаляет дублирующие рамки) и добавляет новые, чтобы нейронная сеть работала быстрее и точнее (рисунок 1.20).

Основные преимущества архитектуры SSD:

- возможность работы в реальном времени;
- качество распознавания близко к Faster R-CNN;
- распознавание происходит на разных масштабах.

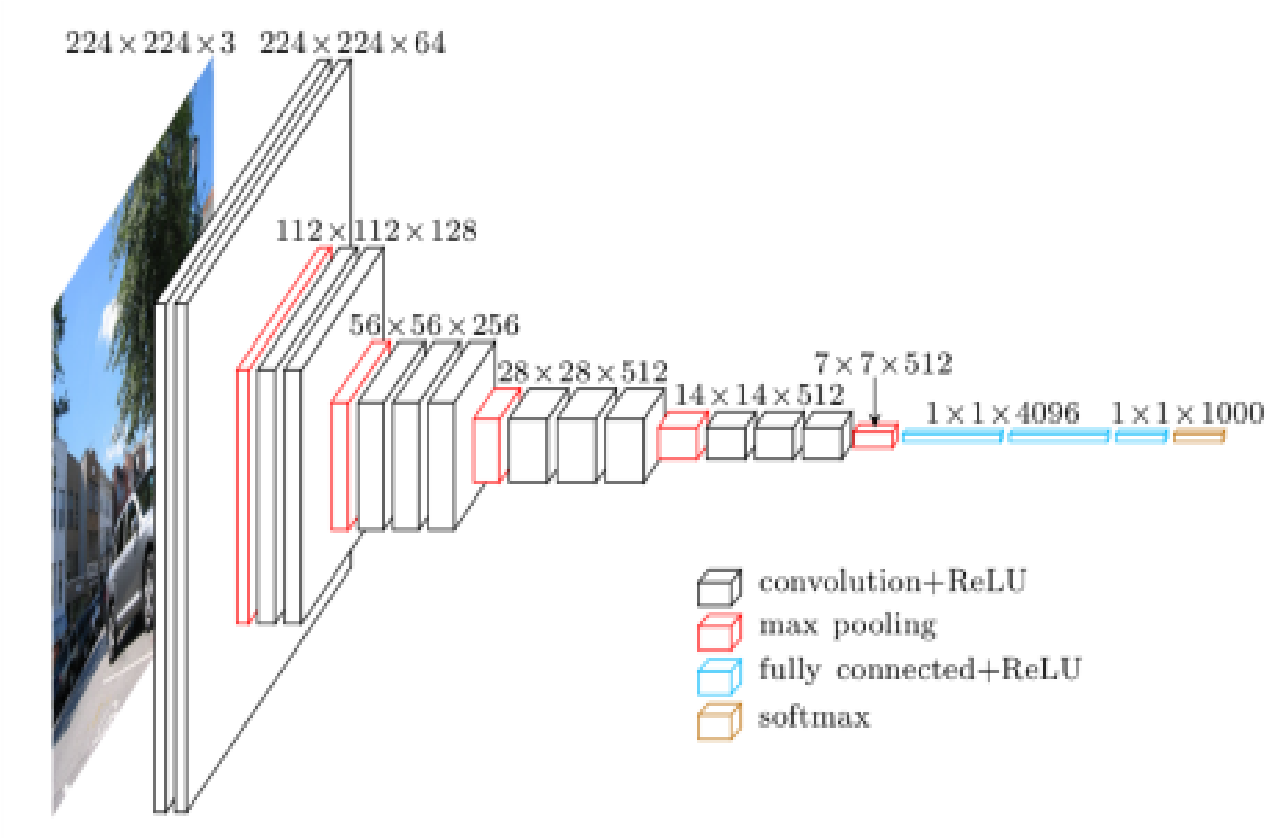

Рисунок 1.20 – Архитектура Single Shot MultiBox detector

Как показали тесты производительности, SSD обрабатывает изображения быстрее, но проигрывает в точности (рисунок 1.21).

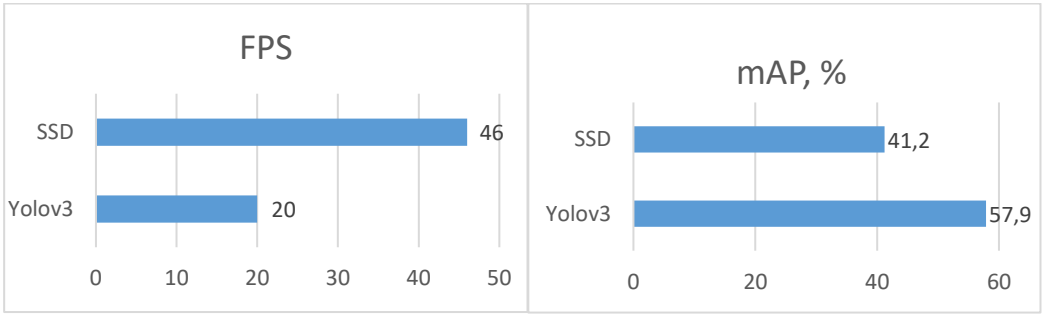

Рисунок 1.21 – Тесты скорость и точности обработки изображения

1.10 Выводы по главе 1

Рассмотрев методы анализа воздуха, которые можно использовать для поставленной задачи определения выбросов от транспортных средств можно сделать вывод о том, что закупка такого оборудования является нецелесообразным в виду их высокой стоимости. Для расчета выбросов был выбран способ расчета, предложенный в ГОСТ Р 56162-2019 с удельными значени-

ями каждого загрязняющего вещества (CO, NO_x, CH, Сажа, SO_2, Формальдегид, Бенз(а)-пирен). Данный метод не требует дополнительного оборудования и относительно легок в реализации. Метод рассчитан на ручной подсчет транспортных средств. В данной работе метод будет автоматизирован, что повышает его эффективность и даст возможность анализировать воздух в области дорожного узла 24 часа в сутки.

Из рассмотренных способов определения скорости ТС был выбран способ определения скорости по видеофайлу. Данный способ определения скорости так же можно применить и в режиме реального времени. Так же, как и анализ выбросов, данный способ не требует дополнительного оборудования.

Рассмотрев архитектуры нейронных сете выбор был между YOLO и SSD. Так как набор данных уже был собран для YOLO, было решено остаются на ней. Данное решение поход по скорости и точности обработки изображения. Последние вышедшие обновления в YOLO v3, на момент написания выпускной квалификационной работы, позволило повысить точность распознавания и классификации.

2 АЛГОРИТМЫ ПОДСЧЕТА ТРАНСПОРТНЫХ СРЕДСТВ ИХ СКОРОСТИ И ВЫБРОСОВ

2.1 Нейронная сеть YOLO v3

2.1.1 Архитектура сети и обучение

Конфигурация нейронной сети YOLOv3 использованной в данной работе имеет 106 свёрточных слоев, остаточные уровни [20], слои повышенной дискретизацией и пропускаемые слои (рисунок 2.1).

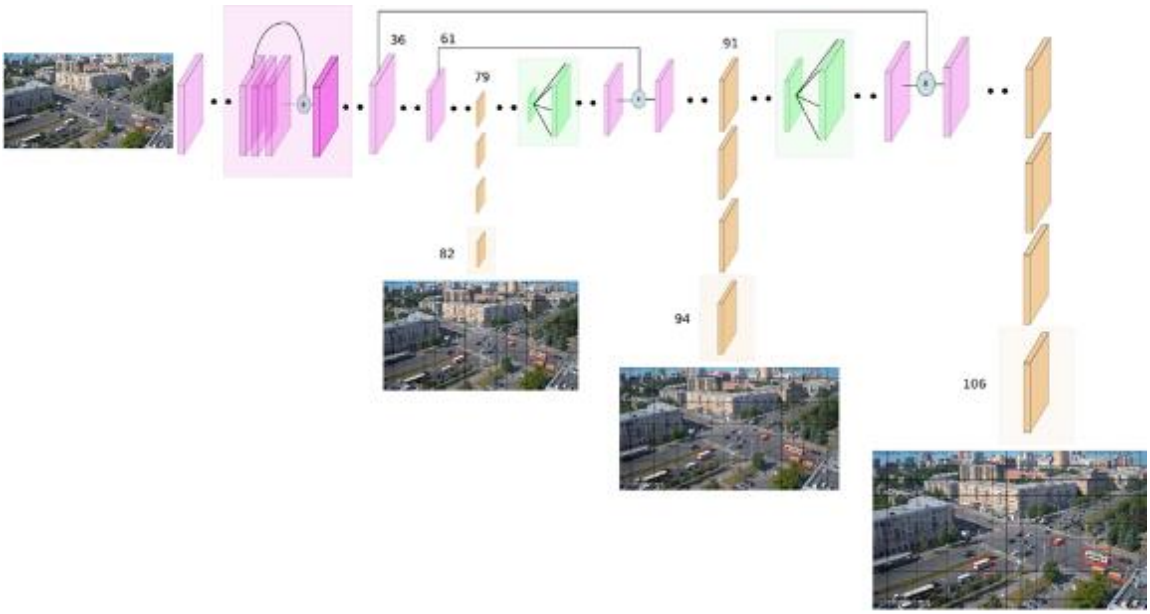

Рисунок 2.1 – Архитектура нейронной сети YOLOv3

На вход CNN получает изображение и возвращает тензор (рисунок 22), который представляет собой:

– b_x, b_y, b_w, b_h – координаты и положения прогнозируемых ограничив ающих рамок, которые должны содержать объекты;

– p_c – вероятность того, что каждый ограничивающий прямоугольник содержит объект;

– C_i – вероятность того, что каждый объект внутри его ограничительн ой рамки принадлежит определенному классу.

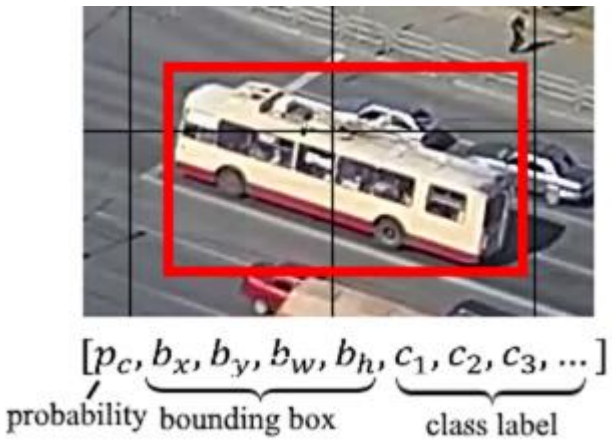

Рисунок 2.2 – Выходной тензор

В качестве данных для обучения нейронной сети мы собрали и разметили кадры видеопотоков с 7-ми камер различных дорожных узлов г. Челябинска. В результате суммарно было получено около 430 000 объектов транспортных средств. Индексация классов и соответствующие им цвета, используемые в дальнейшем для отображения результатов обнаружения, представлены в таблице 1.

Входные данные представлены следующим образом: изображение в формате JPG или PNG и текстовый файл с разметкой:

$$\begin{matrix} C_1 & X_1 & Y_1 & W_1 & H_1 \\ C_2 & X_2 & Y_2 & W_2 & H_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ C_i & X_i & Y_i & W_i & H_i \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ C_n & X_n & Y_n & W_n & H_n, \end{matrix}$$

где i – номер объекта;

n – количество объектов на изображении;

C_i – индекс класса i-ого объекта;

X_i, Y_i – координаты центра прямоугольника, содержащего объект;

W_i, H_i – ширина и высота прямоугольника, содержащего объект.

Параметры $X_i Y_i W_i H_i$ записаны в относительных от размера изображения величинах ($X_i, Y_i, W_i, H_i \in [0; 1]$).

Таблица 1 – Индексация классов и соответствующие им цвета

Индекс	Класс	Цвет рамки
0	Легковой автомобиль	Жёлтый
1	Малый автобус	Синий
2	Средний автобус	Оранжевый
3	Большой автобус	Фиолетовый
4	Малый грузовик	Красный
5	Средний грузовик	Черный
6	Большой грузовик	Фиолетовый
7	Троллейбус	Зелёный
8	Трамвай	Розовый
9	Специальный транспорт	Серый
10	Автопоезд	Голубой

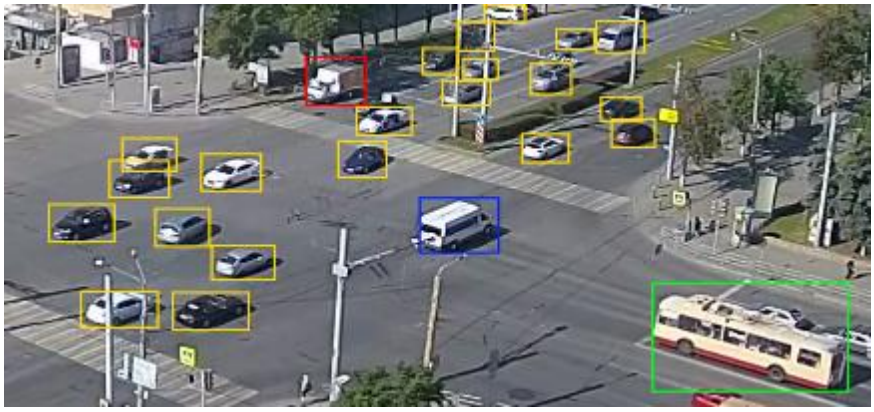

Рисунок 2.3 – Пример размеченного изображения

Для более качественного обучения нейронной сети мы расширили набор данных, применив аугментацию, что увеличило набор данных в 12 раз. Для аугментации были применены следующие преобразования в различных комбинациях: горизонтальное отображение; аффинное и перспективное преобразования; наложение шума; искажение цветов.

YOLO v3 рассматривает изображение один, накладывая на изображение своеобразную сетку. Эта сетка имеет размер $N \times N$. Каждая ячейка данной сетки отвечает за предсказание:

1) Каждая ячейка отвечает за предсказание нескольких ограничительных рамок. Для каждой ограничительной рамки в ячейке предсказывается 4

координаты (b_x, b_y, b_w, b_h), вероятность того, что рамка содержит объект и класс объекта.

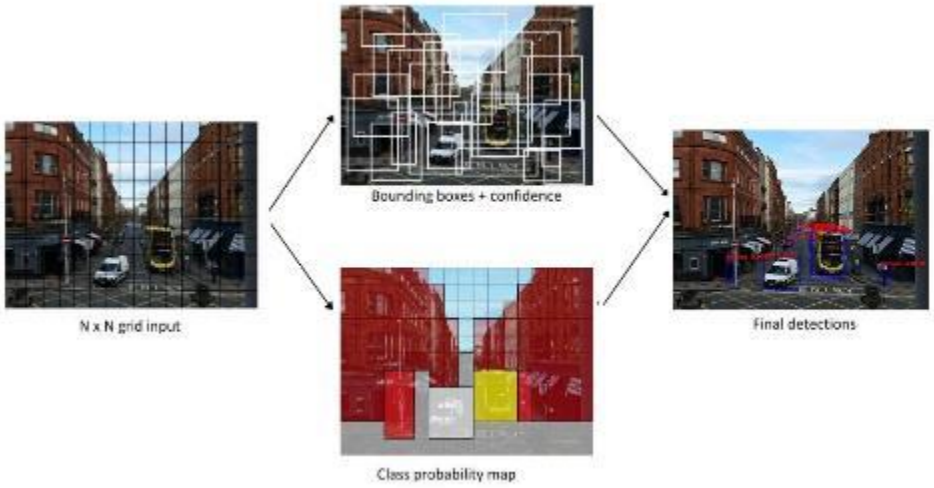

Рисунок 2.4 – Визуализация ограничительных рамок.

2) Каждая ячейка предсказывает вероятности классов. Это не означает, что какая-то ячейка сетки содержит какой-то объект, это просто вероятность. Если ячейка сетки предсказывает автомобиль, это не говорит, что есть автомобиль, это просто говорит, что если есть объект, то этот объект – автомобиль. В общей сложности на выходе получим $1\times 1 \times (B \times (5 * C))$ выходов, где B – количество ограничительных рамок, предсказанных каждой ячейкой (зависит от конфигурации);

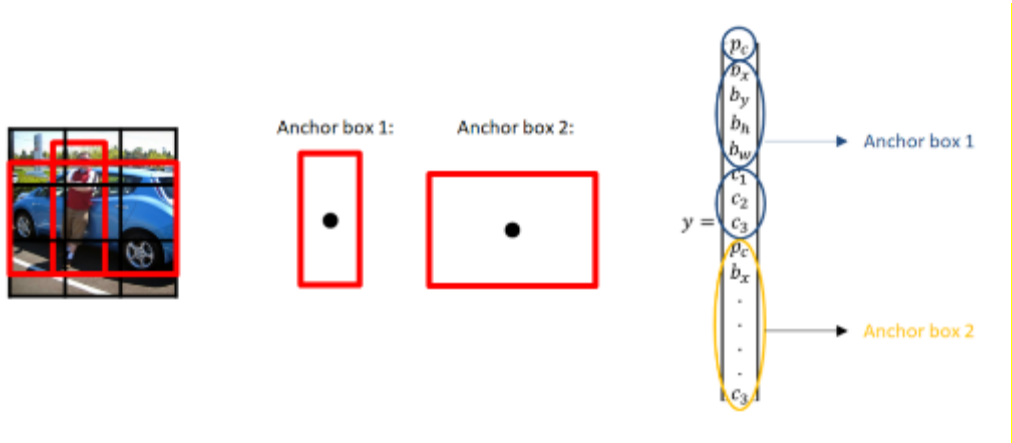

Рисунок 2.5 – Якорные рамки

C – количество классов в обучении;

«5» – координаты ограничительных рамок и вероятность.

За один проход можно перейти от входного изображения к выходному тензору (рисунок 3). Что бы уменьшить количество ограничительных рамок простое пороговое значение, что позволит нам избавится от всех прогнозов с низким значением вероятности. Даже после этого остается много дублирующих рамок. Для удалить дубликаты используется IoU (Intersection Over Union).

На рисунке 2.6 видим 2 рамки, красная является основной, а фиолетовая выходом сети. Как видно из рисунка 5 эти рамки пересекаются не идеально. Поэтому надо измерить, насколько плохим или хорошим является результат сети. Для этого вычисляем IoU.

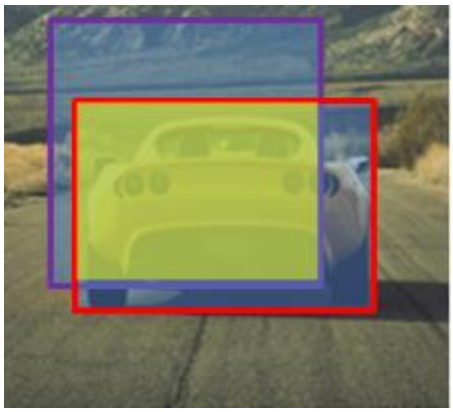

Рисунок 2.6 – Объединение – синий,
желтый – пересечение

В задаче обнаружения объекта наше ожидание заключается в том, чтобы локализовать объект наилучшим из возможных способов. Если красная ограничивающая рамка является основной ограничительной рамкой (где автомобиль находится на изображении), есть обнаружила автомобиль и выводит ограничивающую рамку в фиолетовом цвете IoU говорит нам, есть ли у нас хороший или плохой результат.

Объединение этих двух ограничительных рамок представляет собой синюю область. Это область, которая содержится в обоих ограничительных рамках, в то время как пересечение ящиков – это меньшая желтая область. IoU вычисляет размер пересечения и делит его на размер объединения. По соглашению ограничительная рамка является правильным, если IoU >= 0.5. Если ограничительная рамка, которую мы получили, и истинная ограничи-

тельная рамка идеально перекрывались, то IoU было бы 1, потому что пересечение было бы равно объединению. Пока IoU >= 0.5 полученный ответ можно считать хорошим.

Даже после этой операции могут оставаться дубликаты. Чтобы избавится им от них применяется NMS (Non-Max Suppression).

2.1.2 Функция активации

Функция активации на подобии сигмоидной являются нелинейными что приводит к проблемам с увеличением градиентов и затуханием. Чтобы избежать этих проблем можно использовать более простой вариант – ReLU (rerectified linear unit – выпрямленная линейная функция) (рисунок 2.8 а)

$$f(y) = \max(0, y).$$

Достоинства данной функции:

− отсутствие ресурсоемких операции;

− отсутствие разрастания/затухания градиента;

− быстрое обучение.

Недостатки:

− не всегда надежна, в процессе обучения может «умирать». Например, в случае большого градиента, проходящего через ReLU, привести к такому обновлению весов, что данный нейрон никогда больше не активируется. В таком случае начиная с этого момента, градиент, проходящий через этот нейрон, всегда будет равен нулю;

− сильно зависит от инициализации весов.

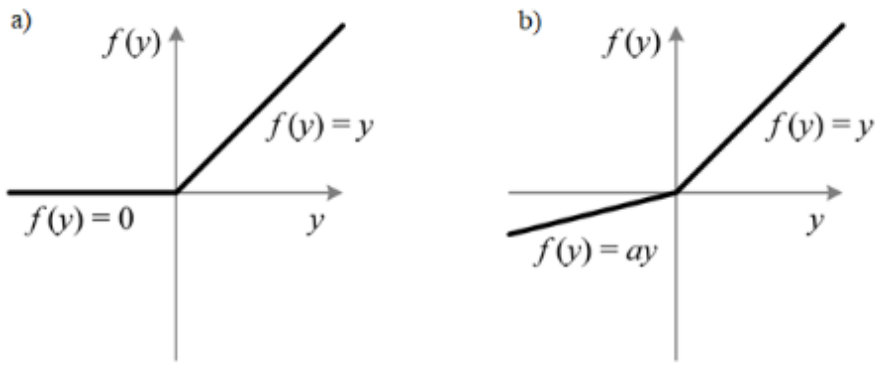

Рисунок 2.7 – а) ReLU, b)Leaky ReLU

Leaky ReLU(leaky rerectified linear unit – выпрямленная линейная функция с «утечкой») одно из решении призванная решить описанные выше проблемы (рисунок 2.8 b). ReLU на интервале $x < 0$ выдает ответ 0, а Leaky ReLU на этом интервале имеет небольшой отрицательный коэффициент.

$$f(x) = \begin{cases} ax \text{ при } x < 0, \\ x \text{ при } x \geq 0. \end{cases}$$

Данная функция активации применяется в нейронной сети YOLO v3.

2.1.3 Результаты обучения

Одним из методов оценки точности классификатора является оценка с помощью метрик precision и recall.

Точностью (precision) называют долю объектов, принадлежащих к данному классу относительно всех объектов которых НС отнесла к этому классу.

Полнота (recall) – это доля найденных классификатором объектов, принадлежащих классу относительно всех объектов этого класса в тестовой выборке.

В таблице 2 содержится информация о том, сколько раз система приняла верное и сколько раз неверное решение по объектам данного класса. А именно:

- TP – правильно распознанный объект;
- FP – неправильно распознанный объект;
- FN – ложное срабатывание.

Таблица 2 – Матрица ошибок

Предсказанный результат	Истинные значения	
	True	False
True	True Positive	False Positive
False	False Negative	True Negative

Точность и полнота вычисляется следующим образом:

$$precision = \frac{TP}{TP + FP}, \quad (1)$$

$$recall = \frac{TP}{TP + FN}. \quad (2)$$

Исходя из формул (1) и (2), была подсчитана точность и полнота определения 6 классов транспортных средств (таблица 3).

Таблица 3 – Показатели точности и полноты, обученной нейросети

Класс	Precision	Recall
Легковой автомобиль	0.96	0.98
Малый автобус	0.81	0.82
Средний автобус	0.79	0.8
Автобус	0.89	0.79
Малый грузовик	0.75	0.83
Средний грузовик	0.79	0.86
Грузовик	0.72	0.65
Троллейбус	0.71	0.7
Спец. Транспорт	0.74	0.88
Трамвай	0.92	0.95
Автопоезд	0.9	0.96

2.2 Подсчет транспортных средств

2.2.1 Постановка задачи

Пусть $X = \{x_i\}$ – это множество объектов $x_i = \{x, y\}$, где x, y – координаты объектов (транспортных средств), полученных от нейронной сети с каждого кадра.

Необходимо сопоставить x_{i-1} и x_i объекты и дать каждому такому уникальному объекту уникальный номер и ведя подсчет.

2.2.2 Теория и алгоритм подсчета ТС

Для подсчета транспортных средств на перекрестке, были выделены зоны, в которых ведется подсчет транспорта (рисунок 2.8). Откуда и куда едет, класс транспорта.

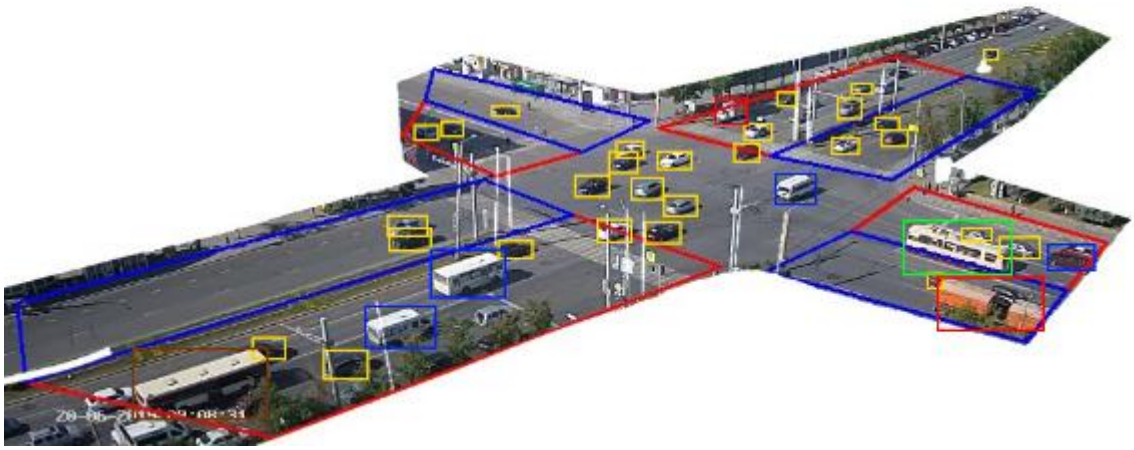

Рисунок 2.8 – Изображение с наложенной маской и визуализация областей въезда и выезда транспортных средств с перекрестка, обозначенных красным и синим цветами, соответственно

Чтобы нейронная сеть не распознавала транспорт за пределами дороги накладываем на каждый кадр маску, закрашивая ненужные области белым цветом.

Для сопоставления и отслеживания транспортных средств в видеопотоке была использована библиотека с открытым исходном кодом Sort [6]. Данная библиотека помогает установить соответствие между транспортом на предыдущем кадре и текущим давая им уникальный номер.

На рисунке 2.9 представлена блок-схема подсчёта ТС по 4-ем направлениям. Для начала получаем координаты объекта и определяем области N, E, S, W. Область $N = \{x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4\}$ содержит точки образующие прямоугольную область (рисунок 2.8). Далее проверяем принадлежит ли точка (x, y) одной из областей N, E, S, W, если принадлежит увеличиваем счетчик данной области на единицу, таким образом ведется подсчет ТС по всем направлениям.

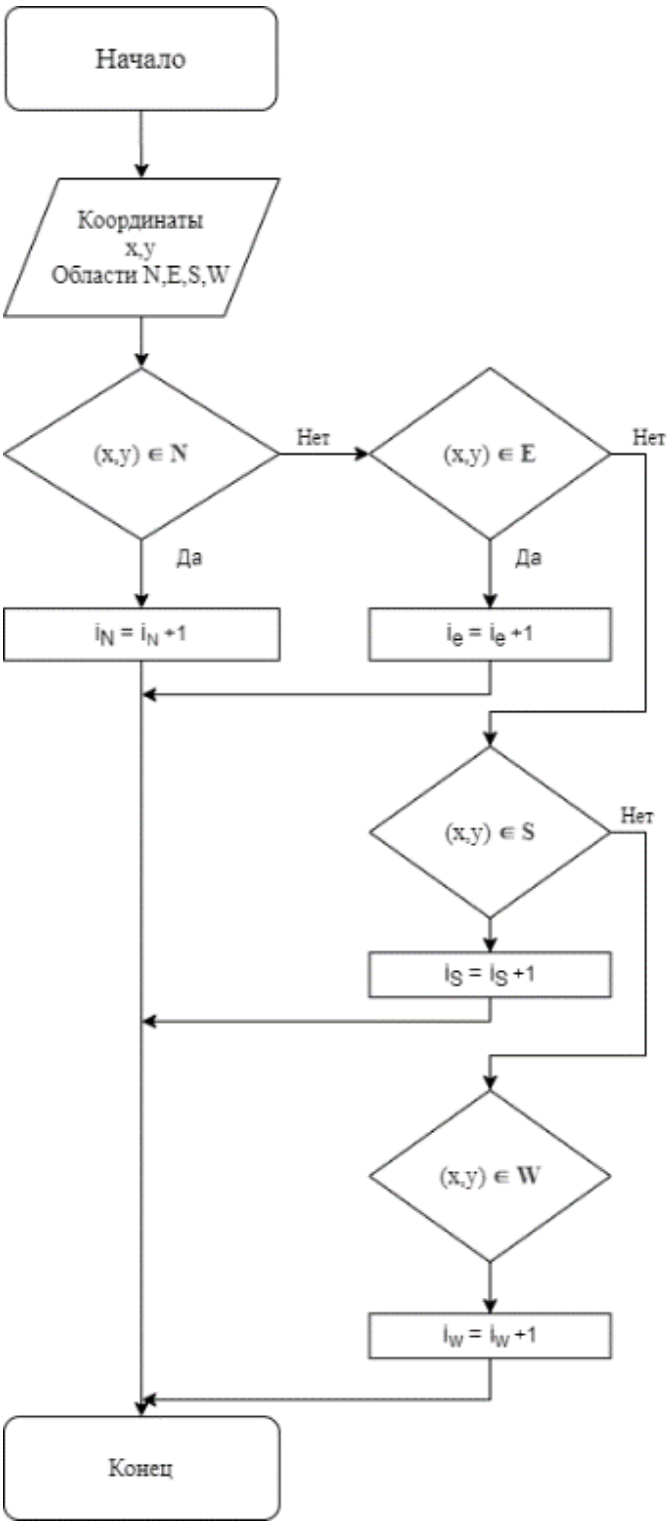

Рисунок 2.9 – Блок схема подсчета ТС

Данный способ очень эффективен так как нейронная сеть может терять ТС на относительно длинное время и при повторном обнаружении будет учитываться в статистике.

2.3 Определение скорости на видео

2.3.1 Постановка задачи

Пусть $X = \{x_i\}$ – множество объектов (набор координат ТС, уникальный номер, координаты (x, y)) $x_i = \{id, x, y, t_0\}$ где x, y – координаты объекта, t_0 – время начала движения, id – уникальный номер объекта.

Необходимо построить алгоритм расчета скорости V для каждого уникального объекта из множества X.

2.3.2 Теория и алгоритм для расчета скорости

Для этого надо узнать изменения широты и долготы ТС за определенный интервал времени, используя изменение координат на изображении с камеры. Для решения этой задачи нужно вычислить матрицу перспективного преобразования (3) выбрав четыре опорные точки на спутниковой карте и сопоставив соответствующие им точки на изображении (рисунок 2.10). Матрица для данного перекрестка выглядит следующим образом:

Рисунок 2.10 – Опорные точки на изображении

$$A = \begin{bmatrix} -30.52765281 & -222.7326644 & 55.16217937 \\ -33.96612068 & -247.85597402 & 61.38201011 \\ -0.5534127 & -4.03790521 & 1.0 \end{bmatrix} \quad (3)$$

Теперь для получения географических координат нужно умножить матрицы:

$$A * \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} = \begin{pmatrix} x_i' \\ y_i' \\ 1 \end{pmatrix}, \quad (4)$$

где A – матрица преобразования,

x_i, y_i – координаты пикселя на изображении

x'_i, y'_i– широта и долгота точки.

Для вычисления расстояния между двумя точками используется библиотека haversine [9], которая принимает широту и долготу точек. Формула, по которой библиотека вычисляет расстояние:

$$d = 2r * \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_2)\cos(\varphi_1)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right), \quad (5)$$

где φ_i, λ_i– широта и долгота i-ой точки,

d – расстояние между двумя точками,

$r =$ радиус земли 6371 км.

Теперь для вычисления средней скорости мы применяем следующую формулу:

$$v = \frac{d}{t_2 - t_1},$$

где t_1, t_2 – время начала и конца движения на дистанции.

У данного способа уже ошибка намного меньше, так как уже нет зависимости от количества кадров, который в случае плохого интернет-соединения могут пропасть.

Проведем теоретическую оценку погрешности вычисления дистанции таким способом. Ошибка вычисления реальных координат с помощью проекции пиксельной карты появляется за счет того, что область, транслируемая одним пикселем на кадре, имеет некоторую ненулевую площадь (рисунок 2.11).

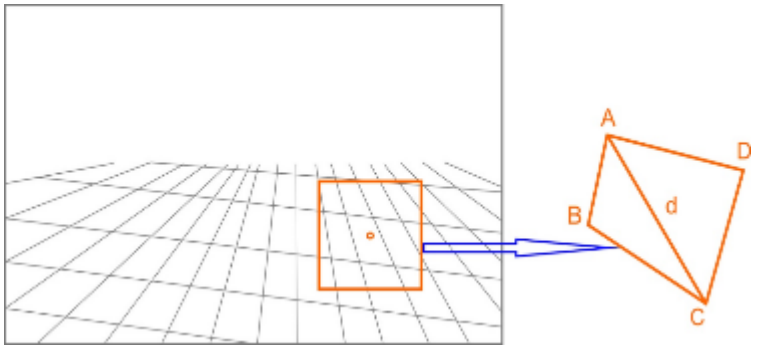

Рисунок 2.11 – Область, транслируемая одним пикселем

Чтобы получить верхнюю оценку этой площади, с учетом перспективы, мы взяли область используемых пикселей, транслирующих самую дальнюю часть дороги (рисунок 2.12). Реальный размер данной дистанции – 88,5m, на изображении этот отрезок транслируется 91px.

Рисунок 2.12 – Дистанция, транслируемая пикселями

Таким образом один пиксель охватывает область длиной 0.97m, приняв эту область за квадрат, можно оценить максимальную погрешность проекции точки в рамках данной области 1.37m. С учетом данной погрешности (рисунке 2.13), ошибка определения дистанции не превышает 2.74m:

$$d - r \leq 2e = 2 \times 1.37m = 2.74m,$$

где r – реальная дистанция,

e – ошибка вычисления координат одной точки.

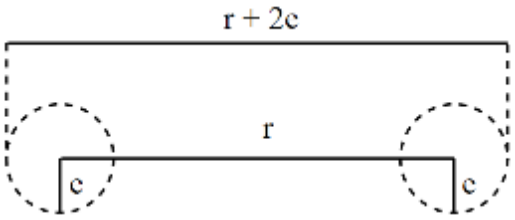

Рисунок 2.13 – Погрешность при измерении дистанции/расстояния

На рисунке 2.14 представлена блок-схема работы алгоритма расчета скорости. Для начала нужно узнать какое количество кадров(frame) отлеживаем данный объект (ТС). Если больше 7 кадров, тогда берем последние 8 кадров из массива кадров и из них получаем координаты с первого и послед-

него кадра и разницу во времени между этими кадрами. Далее считаем расстояние с помощью haversine. Считаем разницу во времени между первым и последним кадром, после чего переводим миллисекунды в часы. На последнем этапе получаем скорость деля расстояние на время.

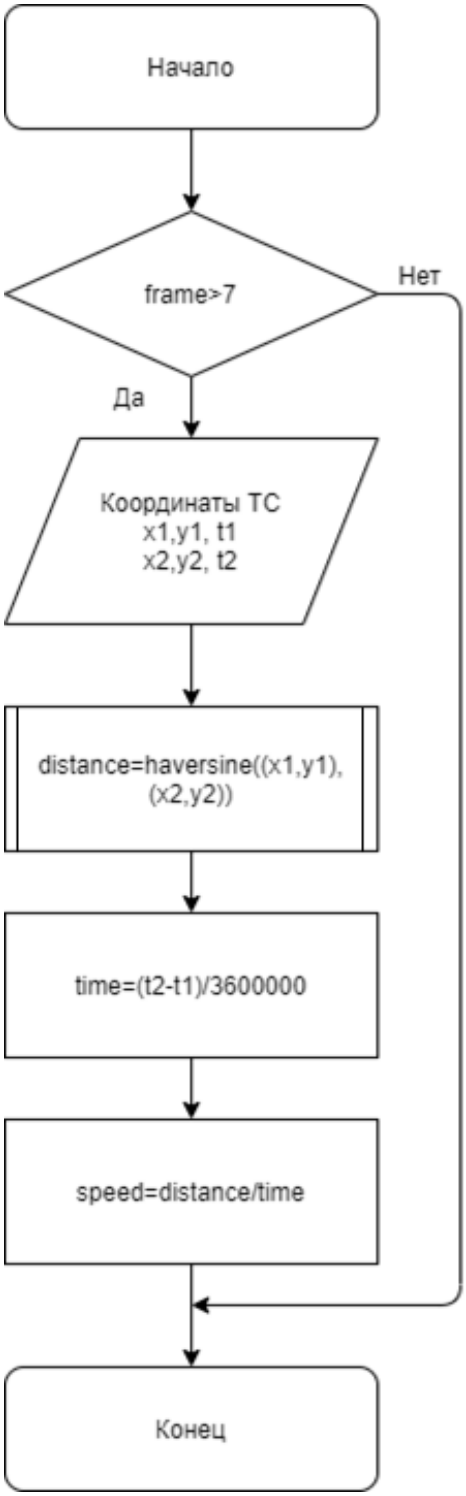

Рисунок 2.14 – Блок схема для расчета скорости ТС

Результаты подсчета скорости транспортных средств представлены на рисунке 2.15.

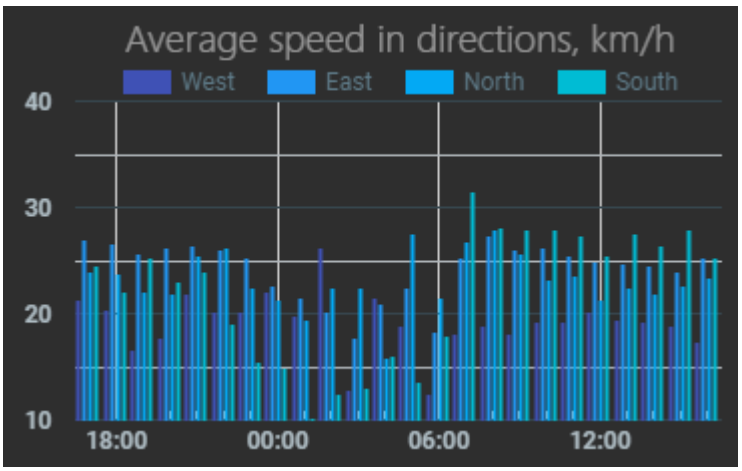

Рисунок 2.15 – График средней скорости по 4-ем направлениям на перекрестке за 24 часа

2.4 Расчет экологических выбросов

2.4.1 Постановка задачи

Пусть $X = \{x_k\}$ – множество уникальных объектов ТС, в котором содержится весь путь x_k = (x, y, класс ТС).

$Y = \{y_{ij}\}$ – матрица с удельными показателями выбросов для 5 классов ТС. Необходимо подсчитать количество ТС для каждой категории:

$$k_i = \sum x_k ,$$

где k_i – количество ТС i-ой категории.

$$E_{ij} = k_i * y_{ij},$$

где E_{ij} – выброс каждого типа выброса для каждого класса ТС;

y_{ij} – среднее значение выброса для k_i класса ТС.

2.4.2 Теория и алгоритм расчета выбросов

Даны способ расчета выбросов [10] использует рассчитанные значения интенсивности и структуры потока за определенный промежуток времени (таблицы 9).

Выброс i-го загрязняющего вещества автомобилями конкретного направления движения в районе перекрестка при запрещающих движение светофора за 20-минутный период обследования $M^3_{П_i}$, г/км, вычисляется по формуле:

$$\mathrm{M}^{3}_{\Pi_i} = \frac{\mathrm{P_Ц}}{60} * \sum_{1}^{N_Ц} \sum_{1}^{k} \left(M'_{\Pi_{i,k}} * G_{k\Pi} \right), \quad (6)$$

где $\mathrm{P_ц}$ – продолжительность действия запрещающего движения сигнала светофора (включая желтый цвет) в течение 20 мин, с;

$N_ц$ – количество циклов действия запрещающего движения сигнала светофора за 20-минутный период времени;

$M'_{\Pi_{i,k}}$ – удельный выброс i-го загрязняющего вещества автомобилями, k-й группы, находящихся в очереди у запрещающего движения сигнала светофора, определяемый по таблице 1, г/мин;

$G_{k\Pi}$ – количество автомобилей k-й группы, находящихся в очереди в районе перекрестка в конце каждого цикла действия запрещающего движение сигнала светофора.

В таблице 4 приведены средние значения удельных выбросов загрязняющих веществ в граммах в минуту, учитывающий движение автомобилей в районе перекрестка.

Таблица 4 – Удельные выбросы для транспортных средств по 5 группам

Наименование группы автомобилей	Номер группы	Выброс, г/мин						
		CO	NO_x (в пересчете на NO_2)	CH	Сажа	SO_2	Формальдегид	Бенз(а)пирен
1	2	3	4	5	6	7	8	9
Легковые	I	0,17	$8{,}80 * 10^{-3}$	0,033	0,011	$0{,}17 * 10^{-2}$	$0{,}26 * 10^{-3}$	$0{,}13 * 10^{-6}$
Автофургоны и микроавтобусы массой до 3,5 т	II	1,00	$30{,}00 * 10^{-3}$	0,070	0,033	$0{,}33 * 10^{-2}$	$0{,}30 * 10^{-3}$	$0{,}13 * 10^{-6}$

Продолжение таблицы 4

1	2	3	4	5	6	7	8	9
Грузовые массой от 3,5 до 12 т	III	1,00	$9{,}90 * 10^{-2}$	0,170	0,220	$0{,}55 * 10^{-2}$	$0{,}76 * 10^{-3}$	$0{,}33 * 10^{-6}$
Грузовые массой свыше 12 т	IV	2,00	$13{,}00 * 10^{-2}$	0,260	0,450	$0{,}66 * 10^{-2}$	$1{,}16 * 10^{-3}$	$0{,}40 * 10^{-6}$
Автобусы массой свыше 3,5 т	V	0,90	$9{,}90 * 10^{-2}$	0,070	0,120	$0{,}60 * 10^{-2}$	$0{,}25 * 10^{-3}$	$0{,}10 * 10^{-6}$

На рисунке 2.16 представлена блок-схема расчета выбросов для легкового автомобиля. Аналогично вычисляются выбросы и по остальным группам транспортных средств представленных в таблица 4. Данный алгоритм принимает на вход массив данных, в котором содержится информация о количестве транспортных средств присутствующих в области перекрестка за 20 мин. промежуток времени. Используя приведенные значения из таблицы 4, вычисляется выброс вредных веществ для каждой группы автомобилей.

После подсчета выбросов по каждой группе полученные значения делятся на объем области перекрестка равному 14720 м^2 (рисунок 2.17) получая значение в г/м^3, что дает понять какое количество вредных веществ находится в области данного узла.

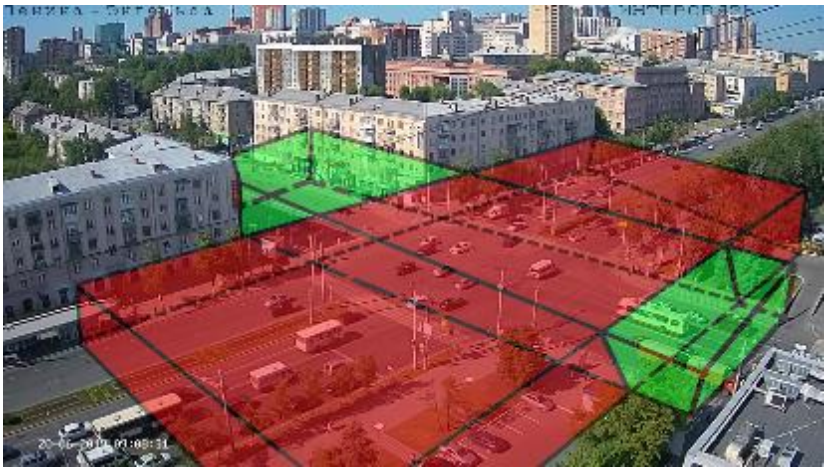

Рисунок 2.16 – Объем области перекрестка

Рисунок 2.17 – Блок схема расчета выбросов

В таблице 5 приведён пример подсчета ТС по направлениям. По таким данным скитаются выбросы. Подсчет ТС критично важен для данного метода расчета выбросов т. к. метод полностью зависит от этих данных. Улучшая точность подсчета ТС, улучается и точность определения выбросов. В таблице 4 представлено только 4 группы ТС, а подсчет ведется по 11 категориям, поэтому все транспортные средства объединяются по критерию веса, указанных в столбце 1 таблицы 4. В таблице 5 показан подсчет только

по 7 категориям т. к. на данном участке в течение часа проезжали только данные типы ТС.

Таблица 5 – Подсчет по категориям на перекрестке Чичерина-Кашириных за 60 минут

	Легковой автомо-биль	Малый авто-бус	Средний автобус	Большой автобус	Малый гру-зовик	Средний грузовик	Большой грузовик
N	798	54	9	1	11	0	0
S	1860	8	1	0	13	1	1
W	730	132	7	4	4	0	1
E	1219	44	24	0	16	0	2
Сумма	4607	238	41	5	44	1	4

2.5 Выводы по главе 2

В данном разделе было рассмотрена теория расчета 3-х алгоритмов: подсчет ТС, расчет скорости и расчет экологических выбросов. Рассмотрена работа нейронной сети YOLO v3, от которой получают данные для работы разработанные алгоритмы.

У разработанных алгоритмов есть один существенный недостаток, вытекающий из способа обнаружения ТС – это помехи, вмещающие обзору камере такие как деревья, рекламные щиты, расположение камеры и т. д. Данные помехи могут мешать обнаружению ТС в следствии чего они не будут учитывается в статистике.

3 ВЫЧИСЛЯЕМЫЕ ПОКАЗАТЕЛИ И ИХ ТОЧНОСТЬ

3.1 Точность подсчета ТС

Для определения точности алгоритма подсчета транспортных средств было проведено ряд тестовых измерении. Используя 10 минутные записи с перекрестков города Челябинска был проведен подсчет транспортных средств вручную и нейронной сетью по 4 четырем направлениям узла соответственно обозначив их по названиям сторон света.

Таблица 6 – Сравнение подсчета ТС на Комсомольский-Ворошилова

	Ручной подсчет	Программа	Абсолютное отклонение	Относительное отклонение
N	143	140	-3	-2%
S	100	104	4	4%
E	182	186	4	2%
W	154	156	2	1%
Сумма	579	586	7	1%

Таблица 7 – Сравнение подсчета ТС на Комсомольский-Ворошилова

	Ручной подсчет	Программа	Абсолютное отклонение	Относительное отклонение
N	-	-	-	-
S	81	83	2	2%
E	121	130	9	7%
W	268	274	6	2%
Сумма	470	487	17	4%

Таблица 8 – Сравнение подсчета ТС на Косарева – Братьев Кашириных

	Ручной подсчет	Программа	Абсолютное отклонение	Относительное отклонение
N	98	98	0	0%
S	207	182	-25	-12%
E	254	247	-7	-3%
W	525	528	3	1%
Сумма	1084	1055	-29	-3%

Таблица 9 – Сравнение подсчета ТС на Победы - Краснознамённая

	Ручной подсчет	Программа	Абсолютное отклонение	Относительное отклонение
1	2	3	4	5
N	136	131	-5	-4%
S	63	70	7	11%

Продолжение таблицы 9

1	2	3	4	5
E	168	169	1	1%
W	312	318	6	2%
Сумма	679	688	9	1%

Таблица 10 – Сравнение подсчета ТС на Чичерина - Победы

	Ручной подсчет	Программа	Абсолютное отклонение	Относительное отклонение
N	361	326	-35	-10%
S	170	170	0	0%
E	190	190	0	0%
W	276	283	7	3%
Сумма	997	969	-28	-3%

Проанализировав полученные результаты (таблица 6–10), можно сказать, что разработанный алгоритм показал себя в данных тестах очень хорошо. Суммарное количество ТС, проехавших по всем направлениям подсчитанное вручную и программой, отличается от 1–4%. Подсчет по каждому направлению на некоторых перекрёстках показал себя не так хорошо (таблица 8 – 9). Такие данные можно объяснить тем, что обзор камере такие помехи как: деревья, рекламные щиты, и т. д. мешающие определить дальнейшее движение ТС.

3.2 Расчет точности расстояния для определения скорости

Для определения точности расчета расстояния с помощью формулы haversine был произведено сравнение с расчетом расстояния с google maps. Использовав спутниковую карту google maps, была определена ширина тротуара 2.13 м. и получены координаты широты и долготы для двух точек $x_1 = (55.160062N, 61.381729E)$, $x_2 = (55.160062N, 61.381763\text{E})$. Данные координаты используются для расчета расстояния формулой haversine, которая показала 2.15 м. получив разницу в расчете – 2 см. что очень хорошим показателем учитывая, что определение расстояния происходит на довольно небольших расстояниях.

Рисунок 3.1 – Ширина тротуара
измеренное с помощью google maps

Беря в расчет показанную точность, можно сказать, что расчет скорости ТС происходит довольно точно. Например, возьмем за расстояние $S = 5$ м, $t = 336$ мс. отсюда получаем скорость $v = 54$ км/ч. Прибавив к расстоянию 2 см. получаем скорость в $v = 56$ км/ч, т. е. теоретически точность определения скорости ± 2км/ч.

3.3 Результаты расчета выбросов вредных веществ

На рисунке 3.2 представлены расчеты выбросов за 24 часа на перекрестке проспект Ленина-Энгельса г. Челябинск. Из графиков выбросов можно проследить явную зависимость выбросов от времени. В результате алгоритм работает в режиме реального времени наблюдая за ситуацией на перекрёсте и обновляя статистику каждый час накопленными данными.

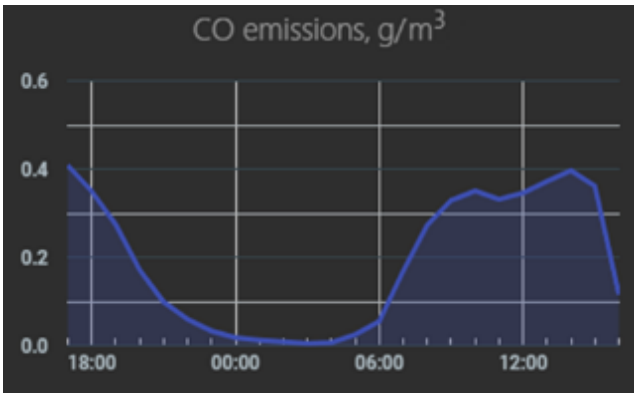

Рисунок 3.2 – Выбросы CO за 24 часа

Изначально метод расчета предполагает ручной подсчет транспорта и последующего вычисления показателей выбросов, но в данной работе этот метод был автоматизирован. Автоматизация таких процессов позволяет повысить эффективность данного метода и позволяет следить за состоянием выбросов 24 часа в сутки. Такой анализ позволяет выявить ошибки проектирования инфраструктуры дорожного узла, такие как слишком долгий запрещающий свет светофора или неправильная организация циклов светофора, что влечет за собой заторы и увеличения выбросов вредных веществ в атмосферу в области перекрестка.

ЗАКЛЮЧЕНИЕ

Цель данной работы состояла в разработке алгоритмов подсчета транспортных средств их скорости и выбросов в режиме реального времени.

Проведя тестовые измерения точности подсчета транспортных средств, показал высокую эффективность, которая укладывается в рекомендованную точность 5%, в случае если обзору камере наружного наблюдения не мешают помехи в виде крон деревьев и рекламных щитов. Данную проблему можно решить в будущем переработав алгоритм подсчета ТС так чтобы не зависеть от областей, показанных на рисунке 2.8.

Теоретическая точность определения скорости транспортного средства равна ± 2 км/ч. Данная точность зависит от точности определения расстояния, который проехал транспорт. Точность определения расстояния зависит от точности определения географических координат. Улучшить данный метод можно рассчитав более точную матрицу перспективного преобразования выбрав для этого другие опорные точки, с помощью которой получаем географические координаты.

Метод определения выбросов от транспортных средств, выбранный в данной работе, применяется для выбросов в ручном режиме. Подсчет ТС для данного метода ведется с помощью людей, стоящих на перекрестке или считающих ТС по записанному заранее видео. Данный метод в данной работе был автоматизирован и ведет подсчет в реальном времени обновляя статистику каждый час суммарным выбросом. Точность определения выбросов целиком зависит от точности подсчета транспортных средств и их классификации.

В результате работы были выполнены все поставленные задачи:

1) разработать алгоритм подсчета транспортных средств по направлениям и категориям (класс ТС, например, легковой автомобиль, автобус, грузовик и т. д.);

2) разработать алгоритм вычисления скорости автомобиля по видеопотоку;

3) разработать алгоритм расчета выбросов от ТС.

Разработанные в результате алгоритмы успешно применяются в интеллектуальной транспортной системе AIMS в рамках проекта «Умный город» проходящее тестирование в режиме реального времени.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Датчик интенсивности «Аркен» [Электронный ресурс] URL: https://www.itc.by/adaptive-control/arken/ (дата обращения 05.10.19)

2 Фреймворк Caffe [Электронный ресурс] URL: http://caffe.berkeleyvision.org/ (дата обращения 13.02.2020)

3 Girshick, R. Rich feature hierarchies for accurate object detection and semantic segmentation/ Jeff Donahue, Trevor Darrell, Jitendra Malik // UC Berkeley [Электронный ресурс] URL: https://arxiv.org/pdf/1311.2524v5.pdf

4 Girshick R. Fast R – CNN / R. Girshick // Microsoft Research [Электронный ресурс] URL: https://arxiv.org/pdf/1504.08083.pdf (дата обращения 02.02.2020)

5 Shaoqing Ren. Faster R-CNN: towards real-time object detection with region proposal networks / Kaiming He Ross Girshick, Jian Sun [Электронный ресурс] URL: https://arxiv.org/pdf/1506.01497v3.pdf (дата обращения 13.02.2020)

6 Bewley A. Simple online and realtime tracking / A. Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, Ben Upcroft // Queensland University of Technology, University of Sydney [Электронный ресурс] URL: https://arxiv.org/pdf/1602.00763.pdf (дата обращения 03.02.2020)

7 Uijlings, J. Selective search for object recognition / J. Uijlings, K. Sande, T. Gevers, A. Smeulders. // International Journal of Computer Vision. 104. (2013).154-171. 10.1007/s11263-013-0620-5.

8 Описание функции Haversine [Электронный ресурс] URL:https://plus.maths.org/content/lost-lovely-haversine (дата обращения 10.02.2020)

9 Zhanga, K. Vehicle emissions in congestion: Comparison of work zone, rush hour and free-flow conditions / K. Zhanga, S. Battermana, F. Dion // Environmental Health Sciences, University of Michigan, Ann Arbor, MI 48109, USA.

10 Girshick, R. Fast R-CNN / Girshick R // 2015, p 1440–8. [Электронный ресурс] URL: https://doi.org/10.1109/ICCV.2015.169.

11 Girshick, R. Rich feature hierarchies for accurate object detection and semantic segmentation / R. Girshick, J. Donahue, T. Darrell, J. Malik // 2012. – P. 580–587, [Электронный ресурс] URL: http://arxiv.org/pdf/1311.2524v3.pdf.

12 He K. Mask R-CNN / K. He, G. Gkioxari, P. Dollar, R. Girshick // 2017, p. 2980–8. IEEE. [Электронный ресурс] URL: https://arxiv.org/pdf/1703.06870.pdf

13 Ren, S. Faster R-CNN: towards real-time object detection with proposal networks / S. Ren, K. He, R. Girshick, J. Sun // 2017;39(6):1137–49. [Электронный ресурс] URL: https://doi.org/10.110 9/TPAMI.2016.2577031.

14 Программное обеспечение Avedex [Электронный ресурс] URL: http://avedex.net/

15 Программное обеспечение TrafficData [Электронный ресурс] URL: https://trafficdata.ru/

16 Сервис определения скорости Car Crash Analysis [Электронный ресурс] URL: https://carcrashanalysis.com/ru/

17 Мобильная станция анализа воздуха [Электронный ресурс] URL: http://www.optec.ru/produktsiya.html?c_dept_id=29&c_good_id=154

18 Классификация газоанализаторов Электронный ресурс] URL: https://www.gazoanalizators.ru/poleznoe.html%26art%3D28

19 Типы газоанализаторов [Электронный ресурс] URL: http://eurolabgas.ru/tipy_gazoanalizatorov

20 He, K. Deep residual learning for image recognition / K He X. Zhang, S. Ren, J. Sun. // 770–78, [Электронный ресурс] URL: https://arxiv.org/pdf/1512.03385.pdf.

21 Liu, W. SSD: Single Shot MultiBox Detector / Wei Liu, D. Anguelov, D. Erhan, Christian Szegedy, S. Reed, Cheng-Yang Fu, A. C. Berg // [Электронный ресурс] URL: https://arxiv.org/pdf/1512.02325.pdf

Приложение 1
Код программы

```
import os
import cv2
from sort import *
import pika
import pickle
import datetime
import time
import json
import threading
import haversine
import datetime
import pandas as pd
from config import *
from shapely.geometry import Point
from shapely.geometry.polygon import Polygon

channel = None
cameras = None

cameras_data = {}

cars_by_time = {}
last_updated_by_time = 0
saving_time = 0
statistic = {
    'cars_by_hour': 0,
    'top_count': 0,
    'left_count': 0,
    'right_count': 0,
    'bottom_count': 0
}
sides = ['left', 'top', 'bottom', 'right']
when_was_start = str(datetime.datetime.now())

predictions = {}
history = pd.DataFrame()
counters = []

to_counter_type = {
    'bike': ['other'],
    'car': ['car'],
    'mini_bus': ['car'],
    'middle_bus': ['public'],
    'bus': ['public'],
    'trolleybus': ['public'],
    'road_train': ['truck'],
    'tram': ['public'],
```

```
    'mini_truck': ['truck'],
    'middle_truck': ['truck'],
    'truck': ['truck'],
    'special_trans': ['other']
}
to_eco_type = {
    'bike': [],
    'car': ['car'],
    'mini_bus': ['mini_bus'],
    'middle_bus': ['bus'],
    'bus': ['bus'],
    'trolleybus': [],
    'road_train': ['truck'],
    'tram': [],
    'mini_truck': ['truck'],
    'middle_truck': ['truck'],
    'truck': ['truck'],
    'special_trans': []
}
side_to_world = {
    'left': 'north',
    'right': 'south',
    'top': 'west',
    'bottom': 'east'
}
cur_kpi = {
    'north': 0,
    'south': 0,
    'west': 0,
    'east': 0
}

def history_init():
    global history, counters
    counters = [
        'count_all',       'count_side_top',       'count_side_right',
'count_side_left', 'count_side_bottom',
        'count_type_car',   'count_type_public',   'count_type_truck',
'count_type_other',
        'count_eco_car',    'count_eco_mini_bus',    'count_eco_bus',
'count_eco_truck',
        'speed_all',    'speed_top',    'speed_right',    'speed_left',
'speed_bottom'
    ]
    history = pd.DataFrame(columns=counters)

history_init()

def history_key(time):
    global history, counters
    key = time.replace(second=0).replace(microsecond=0)
```

```
    if key not in history.index:
        print(history)
        print([0] * len(counters))
        history.loc[key] = [0] * len(counters)
    return key


database = {
    'kpi': {
        'left': 0,
        'right': 0,
        'top': 0,
        'bottom': 0,
    }
}

def database_init():
    global database
    with open('database/db.txt', 'r') as db:
        database = json.loads(db.read())

database_init()

def database_update():
    global database
    with open('database/db.txt', 'w') as db:
        db.write(json.dumps(database))

def save_history(filename=None):
    global history
    if history.empty:
        return
    if filename == None:
        import os
        filename = str(history.index[0]) + '=' + str(history.index[-
1]) + '.json'
        for file in os.listdir('histories/'):
            if str(history.index[0]) == file.split('=')[0]:
                os.rename('histories/' + file, 'histories/' + filename)
                break
    with open('histories/' + filename, 'w') as file:
        json.dump(history.to_json(), file)


def load_history(filename):
    global history
    with open('histories/' + filename) as file:
        history = pd.read_json(json.loads(file.read()))


def load_predictions(filename):
```

```
    global predictions
    with open('history/' + filename) as file:
        predictions = json.load(file)

ecos = ['car', 'mini_bus', 'bus', 'truck']

def eval_eco(history_by_hour):
    global ecos
    emissions = {
        'car': {
            'CO': '0.17',
            'NOx': '0.0088',
            'CH': '0.033',
            'Soot': '0.01',
            'S02': '0.0017',
            'Formaldehyde': '0.00026',
            'Benzopyrene': '0.00000013'},
        'mini_bus': {
            'CO': '1.0',
            'NOx': '0.03',
            'CH': '0.07',
            'Soot': '0.0033',
            'S02': '0.0033',
            'Formaldehyde': '0.003',
            'Benzopyrene': '0.00000013'},
        'bus': {
            'CO': '0.9',
            'NOx': '0.099',
            'CH': '0.07',
            'Soot': '0.12',
            'S02': '0.006',
            'Formaldehyde': '0.00025',
            'Benzopyrene': '0.0000001'},
        'truck': {
            'CO': '1',
            'NOx': '0.099',
            'CH': '0.17',
            'Soot': '0.22',
            'S02': '0.0055',
            'Formaldehyde': '0.00076',
            'Benzopyrene': '0.00000033'}
    }
    gases = ['CO', 'NOx', 'CH', 'Soot', 'S02', 'Formaldehyde',
'Benzopyrene']
    p = 12   #
    t = 0.667  # в секунды
    q = 14720  # кубатура узла
    w = 0.068  # проезжает ТС в км
    m = 1000  # км -> м
    eco_df = pd.DataFrame(columns=['eco_' + i for i in gases],
index=history_by_hour.index).fillna(0)
```

```
    for eco in ecos:
        eco_df['eco_CO_' + eco] = \
            w   *   m   *   (history_by_hour['count_eco_'   +   eco]   *
float(emissions[eco]['CO']) * p * t / 60) / q

    for gas in gases:
        for eco in ecos:
            eco_df['eco_' + gas] += \
                w  *  m  *  (history_by_hour['count_eco_'  +  eco]  *
float(emissions[eco][gas]) * p * t / 60) / q

    max_values = dict()
    for gas in gases:
         max_values[gas] = eco_df['eco_' + gas].max()

    return       {'max_values':       max_values,       'statistic':
json.loads(eco_df.T.to_json(date_format='iso'))}


def count_by_sides(history_by_hour):
    """
    Количество машин по направлениям (в час)
    :param history_by_hour: история, сгруппированная по часам
    :return: датафрейм
    """
    return history_by_hour[[c for c in history_by_hour if 'count_side_'
in c]]

def avg_speeds_by_sides(history_by_hour):
    """
    Средние скорости по направениям (в час)
    :param history_by_hour: история, сгруппированная по часам
    :return: датафрейм
    """
    global sides
    avg_speed_df = pd.DataFrame()
    for side in sides:
        avg_speed_df['avg_speed_' + side] = history_by_hour['speed_' +
side] / history_by_hour['count_side_' + side]
    return avg_speed_df

def count_by_types(history_by_hour):
    """
    Количество автотранспорта по типам (за последние сутки)
    :param history_by_hour: история, сгруппированная по часам
    :return: датафрейм
    """
    return history_by_hour[[c for c in history_by_hour if 'count_type_'
in c]].sum()
```

```
def eval_congestion(history_by_hour):
    """
    Загруженность дорожного узла
    :param history_by_hour: история, сгруппированная по часам
    :return: датафрейм
    """
    global sides
    congestion_df = pd.DataFrame()
    for side in sides:
        congestion_df['congestion_' + side] = history_by_hour['count_side_' + side].tail(1) / 4000  # 4000 - optimal congestion
    return congestion_df

def eval_kpi(history_by_hour):
    """
    KPI дорожного узла
    :param history_by_hour: история, сгруппированная по часам
    :return: датафрейм
    """
    global sides, database
    kpi_df = pd.DataFrame()
    for side in sides:
        side_kpi = history_by_hour['count_side_' + side].max() / 4000 # 4000 - optimal congestion
        if database['kpi'][side] < side_kpi:
            database['kpi'][side] = side_kpi
        kpi_df['kpi_' + side] = [database['kpi'][side]]
    return kpi_df.sum()


def eval_max_vehicles(history_by_hour):
    """
    Максимальное количество автомобилей за час
    :param history_by_hour: история, сгруппированная по часам
    :return: датафрейм
    """
    return 0 if history.empty else str(history_by_hour['count_all'].max())


def current_count(history):
    """
    Максимальное количество автомобилей за час
    :param history: история, сгруппированная по часам
    :return: датафрейм
    """
    return 0 if history.empty else str(history.iloc[-1]['count_all'])


hour_metrics = {
```

```
    'pandas': [
        {'name': 'count_by_sides', 'handler': count_by_sides},
        {'name':           'avg_speeds_by_sides',           'handler':
avg_speeds_by_sides},
        {'name': 'count_by_types', 'handler': count_by_types},
        {'name': 'congestion', 'handler': eval_congestion},
        {'name': 'kpi', 'handler': eval_kpi},
    ],
    'json': [
        {'name': 'eco_values', 'handler': eval_eco},
        {'name':           'max_vehicles_in_hour',           'handler':
eval_max_vehicles},
    ]
}
metrics = {
    'pandas': [],
    'json': [
        {'name': 'current_count', 'handler': current_count},
    ]
}


def callback(ch, method, properties, body):
    global cameras_data, cameras, cars_by_time, last_updated_by_time,
cars_by_hour, saving_time, predictions
    global   counters,   history,   to_counter_type,   hour_metrics,
to_eco_type

    data = json.loads(body.decode())
    millis = data["millis"]
    detections = data["detections"]
    camera_id = data["cameraId"]
    camera = cameras[camera_id]
    anchor_points = camera["anchorPoints"]
    areas = camera["areas"]
    if camera_id in cameras_data:
        camera_data = cameras_data[camera_id]
    else:
        cameras_data[camera_id] = camera_data = {}
    if "sort_tracker" in camera_data:
        sort_tracker = camera_data["sort_tracker"]
    else:
        camera_data["sort_tracker"] = sort_tracker = Sort()
    if "cars_tracks" in camera_data:
        cars_tracks = camera_data["cars_tracks"]
    else:
        camera_data["cars_tracks"] = cars_tracks = {}

    det = []
    for detection in detections:
```

```
        det.append([detection[0],     detection[1],     detection[2],
detection[3], detection[4]])

    trackers = sort_tracker.update(np.array(det))

    filtered_detections = []

    for tracker in trackers:
        x1, y1, x2, y2 = tracker[0], tracker[1], tracker[2], tracker[3]
        cx1 = x1 + (x2 - x1) / 2
        cy1 = y1 + (y2 - y1) / 2

        min_detection = None
        min_dist = -1.0
        for detection in detections:
            cx2 = detection[0] + (detection[2] - detection[0]) / 2
            cy2 = detection[1] + (detection[3] - detection[1]) / 2
            dist = ((cx2 - cx1) ** 2 + (cy2 - cy1) ** 2) ** 0.5
            if min_dist == -1 or dist < min_dist:
                min_dist = dist
                min_detection = detection

        if min_detection is not None:
            det = {
                "id": int(tracker[4]),
                "x1": x1,
                "y1": y1,
                "x2": x2,
                "y2": y2,
                "score": min_detection[4],
                "type": min_detection[5]
            }
            filtered_detections.append(det)

    # print(filtered_detections)

    src = np.array((
        (anchor_points[0]["x"], anchor_points[0]["y"]),
        (anchor_points[1]["x"], anchor_points[1]["y"]),
        (anchor_points[2]["x"], anchor_points[2]["y"]),
        (anchor_points[3]["x"], anchor_points[3]["y"])
    ), dtype=np.float32)
    dest = np.array((
        (anchor_points[0]["lat"], anchor_points[0]["lng"]),
        (anchor_points[1]["lat"], anchor_points[1]["lng"]),
        (anchor_points[2]["lat"], anchor_points[2]["lng"]),
        (anchor_points[3]["lat"], anchor_points[3]["lng"])
    ), dtype=np.float32)

    mtx = cv2.getPerspectiveTransform(src, dest)
```

```
    for detection in filtered_detections:
        x1,   y1,   x2,   y2   =   detection["x1"],   detection["y1"],
detection["x2"], detection["y2"]
        cx1 = x1 + (x2 - x1) / 2
        cy1 = y1 + (y2 - y1) / 2

        original    =    np.array([((0.7,    0.4),    (cx1,    cy1))],
dtype=np.float32)
        converted = cv2.perspectiveTransform(original, mtx)

        detection["lat"] = float(converted[0][1][0])
        detection["lng"] = float(converted[0][1][1])

    for detection in filtered_detections:
        x1,   y1,   x2,   y2   =   detection["x1"],   detection["y1"],
detection["x2"], detection["y2"]
        cx = x1 + (x2 - x1) / 2
        cy = y1 + (y2 - y1) / 2
        point = Point(cx, cy)

        for area in areas:
            points = []
            for p in area["in"]:
                points.append((p["x"], p["y"]))
            polygon = Polygon(points)
            if polygon.contains(point):
                detection["zone"] = area["description"] + " in"
                detection["zone_side"] = area["description"]
                detection["zone_direction"] = "in"
            points = []
            for p in area["out"]:
                points.append((p["x"], p["y"]))
            polygon = Polygon(points)
            if polygon.contains(point):
                detection["zone"] = area["description"] + " out"
                detection["zone_side"] = area["description"]
                detection["zone_direction"] = "out"

    for detection in filtered_detections:
        car_id = detection["id"]
        if car_id in cars_tracks:
            car_tracks = cars_tracks[car_id]
        else:
            cars_tracks[car_id] = car_tracks = []

        car_track = detection.copy()
        car_track["millis"] = millis
        car_tracks.append(car_track)

        if len(car_tracks) > 7:
```

```
            lat1, lng1, millis1 = car_tracks[-8]["lat"], car_tracks[-
8]["lng"], car_tracks[-8]["millis"]
            lat2, lng2, millis2 = car_tracks[-1]["lat"], car_tracks[-
1]["lng"], car_tracks[-1]["millis"]
            dist = haversine.haversine((lat1, lng1), (lat2, lng2))
            dt = (millis2 - millis1) / 3600000
            car_tracks[-1]["speed"] = dist / dt
            detection["speed"] = dist / dt

    data["detections"] = filtered_detections

    channel.basic_publish(exchange="",          routing_key="result",
body=json.dumps(data))

    to_site = dict()
    to_site["detections"] = filtered_detections

    now   =   int(int(round(time.time()   *   1000))   /   1000)     #
round(time.time())
    new_cars = []
    if now - saving_time > 120:
        saving_time = now
        save_history()

    if now - last_updated_by_time < 5:
        channel.basic_publish(exchange="",     routing_key="result2",
body=json.dumps(to_site))
        return

    last_updated_by_time = now
    for car_id in list(cars_tracks):
        car_tracks = cars_tracks[car_id]
        last_car_track = car_tracks[-1]
        seconds = int(last_car_track["millis"] / 1000)
        if now - seconds > 30:
            del cars_tracks[car_id]

            zone_from = None
            zone_to = None
            for car_track in car_tracks:
                if "zone_direction" in car_track and "zone_side" in
car_track:
                    if car_track["zone_direction"] == "in":
                        zone_from = car_track["zone_side"]
                    if car_track["zone_direction"] == "out":
                        zone_to = car_track["zone_side"]

            if zone_from is not None and zone_to is not None:
                sum_speed = 0.0
                frame_with_speed_count = 0
                for car_track in car_tracks:
```

```
                    if "speed" in car_track:
                        speed = car_track["speed"]
                        sum_speed += speed
                        frame_with_speed_count += 1

                if frame_with_speed_count > 0:
                    speed = sum_speed / frame_with_speed_count
                else:
                    speed = 0.0

                new_cars.append({
                    "seconds": seconds,
                    "speed": speed,
                    "from": zone_from,
                    "to": zone_to,
                    "type": last_car_track["type"]
                })

    for new_car in new_cars:
        time_mark                                                    =
datetime.datetime.fromtimestamp(new_car["seconds"])
        key = history_key(time_mark)
        history.loc[key, 'count_all'] += 1
        history.loc[key, 'speed_all'] += new_car['speed']

        for counter_type in to_counter_type[new_car['type']]:
            history.loc[key, 'count_type_' + counter_type] += 1

        for eco_type in to_eco_type[new_car['type']]:
            history.loc[key, 'count_eco_' + eco_type] += 1

        if new_car['to'] is not None:
            history.loc[key, 'count_side_' + new_car['to']] += 1
            history.loc[key,    'speed_'    +    new_car['to']]    +=
new_car['speed']

    history_by_hour     =     history     if     history.empty     else
history.resample('H').sum()

    to_site['cars_info'] = {}

    for metric in metrics['pandas']:
        to_site['cars_info'][metric['name']]                         =
json.loads(metric['handler'](history)

.T.to_json(date_format='iso'))

    for metric in metrics['json']:
        to_site['cars_info'][metric['name']]                         =
metric['handler'](history)
```

```
    for metric in hour_metrics['pandas']:
        to_site['cars_info'][metric['name']]                          =
json.loads(metric['handler'](history_by_hour)

.T.to_json(date_format='iso'))

    for metric in hour_metrics['json']:
        to_site['cars_info'][metric['name']]                          =
metric['handler'](history_by_hour)

    print('history: \n', history)

    channel.basic_publish(exchange="",          routing_key="result2",
body=json.dumps(to_site))

def consumer():
    global channel
    channel.start_consuming()

def main():
    global channel, cameras

    cameras = open_config_dict()

    load_predictions('leneng14-23.json')

    connection                                                        =
pika.BlockingConnection(pika.ConnectionParameters('localhost'))
    channel = connection.channel()
    channel.queue_declare(queue='result',   arguments={"x-max-length":
10}) #video translator
    channel.queue_declare(queue='result2',  arguments={"x-max-length":
10}) #site

    channel.queue_declare(queue='neural',   arguments={"x-max-length":
100})
    channel.basic_consume(queue='neural',               auto_ack=True,
on_message_callback=callback)

    thread = threading.Thread(target=consumer)
    thread.daemon = True
    thread.start()

    input()


if __name__ == "__main__":
    main()
```