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: 01.03.02 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.

Мониторинг транспортных средств по всем направлениям дорожного узла в видеопотоке реального времени с помощью нейронной сети YOLOv3

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ ЮУрГУ–
01.03.02.2020.042.ПЗ ВКР

Руководитель работы, к.ф.-м.н.,
доцент кафедры ПМиП
_____/Т.В. Карпета
«____»_____2020 г.
Автор работы
Студент группы ЕТ-412
_____/К.В. Хазюков
«____»_____2020 г.
Нормоконтролер,
ст. преподаватель
_____/Н.С. Мидоночева
«____»_____2020 г.

Челябинск
2020

АННОТАЦИЯ

Хазюков К.В. Мониторинг транспортных средств по всем направлениям дорожного узла в видеопотоке реального времени с помощью нейронной сети YOLOv3. – Челябинск: ЮУрГУ, ЕТ-412, 73 с., 40 ил., 4 табл., библиогр. список – 30 наим.

Целью данной работы является реализация системы сбора данных о структуре транспортного потока и направлениях движения транспортных средств в режиме реального времени. Решается задача распознавания транспортных средств на изображениях, получаемых с видеопотока уличной камеры. В качестве решения данной задачи применяется нейросетевой подход с использованием архитектуры YOLOv3.

Для программной реализации предложенного решения используется язык программирования Python с применением библиотек Darknet и OpenCV. Для подготовки и обработки данных используются пакеты Numpy и Pandas. Анализ эффективности программы проведен на тестовых данных, полученных с видеопотока уличной камеры.

Результаты данной работы могут быть использованы в интеллектуальных транспортных системах.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Урбанизация и увеличенная плотность застройки городов – существенные особенности современного общества. Поэтому одна из ключевых проблем современного общества – анализ дорожного движения и его эффективная организация. Высокая плотность городского населения приводит к огромному числу личных автомобилей, увеличенному числу грузовых транспортных средств. Это влечет за собой появление дорожных заторов и снижает эффективность использования дорожно-транспортной сети. Для желаемого результата задачи транспортировки больше не могут обращаться к эвристикам, основанным на небольшом количестве вручную собранной статистики [6].

На сегодня во многих городах установлены центры телеавтоматического управления движением трафика, в которых специалисты занимаются задачами корректировки работы светофоров, полагаясь на количество транспорта. Но для того, чтобы принимать эффективные решения по оптимизации дорожного движения, необходимо предсказать и оценить его последствия, при этом опираясь на состояние всей дорожной системы с целым набором показателей. Это значит, что задачи транспортной логистики, в наше время, требуют работы с большими данными. В условиях инфраструктурных ограничений городов и дефицита бюджета, эффективное увеличение пропускной способности дорожной сети возможно обеспечить внедрением интеллектуальных транспортных систем (ИТС). ИТС позволяют анализировать и повышать эффективность использования дорожно-транспортной инфраструктуры и транспортно-эксплуатационные показатели трафика. Основой ИТС являются большие данные (big data), которые необходимо получать, интерпретировать и использовать в режиме реального времени. В этом контексте важнейшей задачей в исследованиях транспортных систем, как науки, становится разработка технологий и стандартов сбора информации с использованием суще-

ствующих коммуникационных сетей на основе использования современной аналитики данных [4].

Данная работа посвящена проблеме получения качественных данных о параметрах дорожного трафика в режиме реального времени, на основе данных с камер видеонаблюдения. Разработка системы по сбору, интерпретации и анализу данных о транспортных потоках в режиме реального времени, на основе использования существующей уличной инфраструктуры, позволит снизить капитальные и эксплуатационные затраты, уменьшить загруженность дорог и эмиссию выхлопных газов, повысить эффективность функционирования транспортных систем.

Цель работы – разработать систему сбора данных о структуре транспортного потока, направлениях движения и количестве транспортных средств в режиме реального времени. Для достижения поставленной цели необходимо решить следующие задачи:

1) исследовать существующие в мире подходы для мониторинга дорожного движения;

2) выбрать подход к решению задачи мониторинга ДД;

3) проанализировать и подготовить исходные данные для обучения нейронной сети;

4) разработать подходящую модель нейронной сети;

5) обучить НС и разработать необходимые программные модули;

6) протестировать работу системы в режиме реального времени.

1 ТЕХНОЛОГИИ МОНИТОРИНГА ТРАНСПОРТНЫХ СРЕДСТВ

1.1 Существующие решения

1.1.1 Система Sensys

Sensys – система определения интенсивности потока транспортных средств (рисунок 1.1), от компании «ЕвроМобайл». Состоит из беспроводных магнитно-резистивных датчиков, предназначенных для обнаружения транспортных средств. Датчик устанавливается в дорожное покрытие проезжей части и передает данные обнаружения в реальном времени.

Рисунок 1.1 – Система определения интенсивности потока транспортных средств Sensys

При необходимости набор компонентов можно расширить. Например, при наличии рядом с автотрассой помех распространения радиосигнала, возможна установка дополнительных антенн и ретрансляторов, для поддержки надежного постоянного сигнала. В качестве полезных свойств данного решения можно выделить:

1) гибкость. Система определения интенсивности потока транспортных средств может быть развернута как на дорогах с малой интенсивностью, так и скоростных многополосных магистралях, а также в тоннелях и на мостах.

При потере сигнала от дорожного датчика все данные записываются в его внутреннюю память, а при восстановлении связи полученная информация о событиях снова начинает передаваться на сервер;

2) автономность. Дорожный датчик работает от встроенного аккумулятора и способен производить до 300 миллионов обнаружений, работая в суровых условиях окружающей среды;

3) удобная диспетчеризация. Все датчики могут заводиться в единый мониторинговый центр и при помощи удобного веб-интерфейса выводить статистику и контролировать сбор данных от датчиков.

Несмотря на достоинства, система Sensys обладает набором технических и иных недостатков, который делает решение задач определенного назначения затратным и трудоемким. Из самых значимых можно выделить следующие:

1) малый радиус обнаружения. Датчики VSN240-F-GR и VSN240-F-2, используемые в данной системе относятся к датчикам типа MAG2/GR, имеющий радиус обнаружения до 3-х метров;

2) большой комплект оборудования. В набор оборудования входит 2 типа датчиков (дополнительный для обнаружения малогабаритных транспортных средств), контроллер, точка доступа, 2 типа ретрансляторов (для организации связи между беспроводными датчиками и точками доступа на разных расстояниях) и блок защиты от короткого замыкания.

Таким образом, для охвата большой площади необходим огромный набор устройств. Поэтому использовать данное решение для полного мониторинга перекрестка или целой сети транспортных узлов довольно трудоемко и затратно.

1.1.2 Программное обеспечение TrafficData

TrafficData – это программное обеспечение, в котором реализованы технологии компьютерного зрения с применением архитектур сверточных нейронных сетей [1]. Данное решение (рисунок 1.2) позволяет обработать за-

груженный видеофайл с изображением дорожного узла. ПО ведет подсчет и классифицирует ТС в соответствии с приказом «Об утверждении Порядка мониторинга дорожного движения».

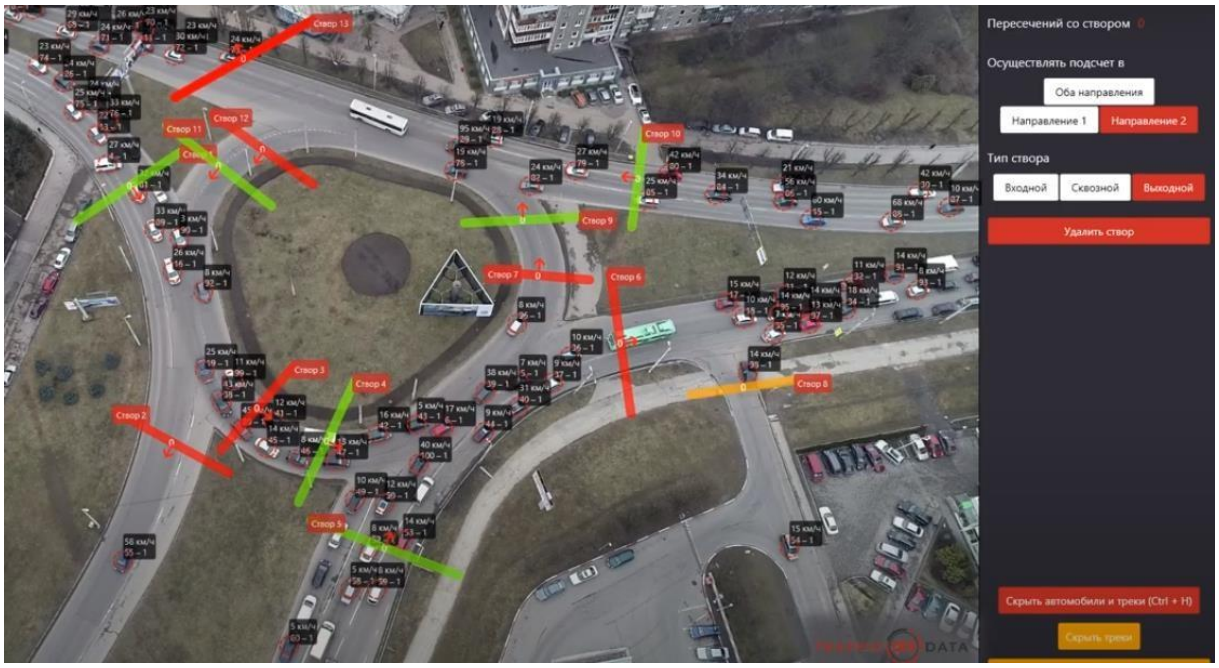

Рисунок 1.2 – Программное обеспечение Traffic Data

Перед обработкой видео пользователю необходимо обозначить отрезки, при пересечении которых будет происходить подсчет ТС. Результаты работы программы можно выгрузить в формате *.XLS для дальнейшего анализа пользователем. В качестве плюсов данного решения можно выделить:

1) точность обнаружения на сложных узлах. Как видно из рисунка 1.2 программа способна обрабатывать большие и сложные участки дорожно-транспортной сети;

2) разнообразие типов транспорта. Traffic Data способен определять не только легковые автомобили, но также грузовой транспорт разных категорий, автобусы, троллейбусы и т. д.

Одним из главных недостатков ПО является отсутствие возможности обработки данных в режиме реального времени, что делает невозможным решение ряда современных задач, стоящих перед транспортной теорией и системами автоматического управления транспортными потоками.

1.1.3 Программное обеспечение Avedex

Программное обеспечение Avedex (рисунок 1.3) предназначено для автоматического анализа автомобильного трафика по видеоизображению.

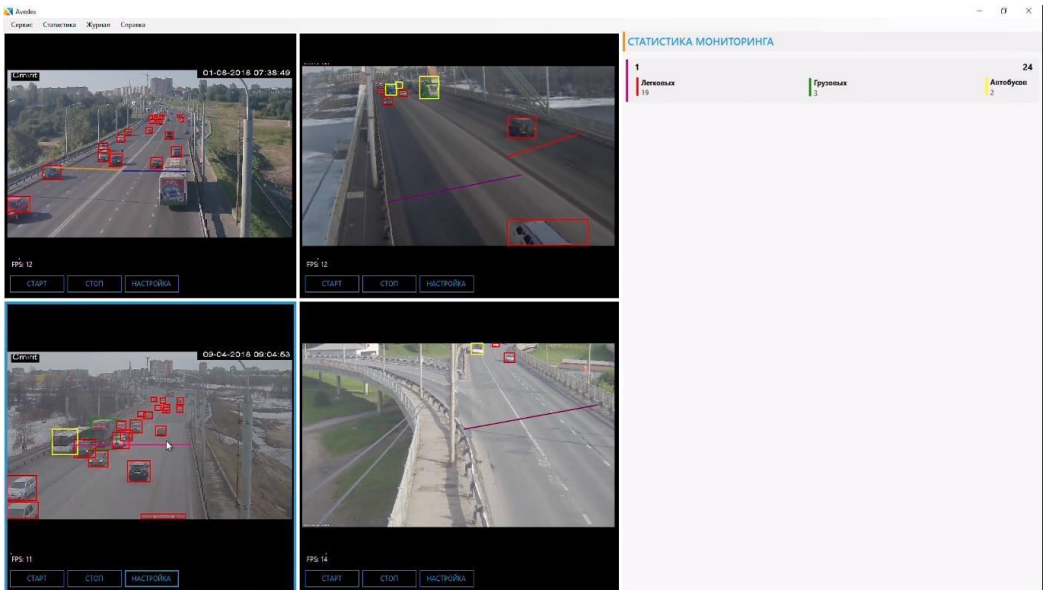

Рисунок 1.3 – Программное обеспечение Avedex

Данное программное обеспечение распознает три типа транспортных средств: легковые автомобили; грузовики; автобусы. Одним из самых важных достоинств Avedex является способность обрабатывать видеопотоки в режиме реального времени. Но также можно выделить ряд существенных недостатков данного решения:

– малое количество классов транспортных средств;

– низкая точность распознавания при сложных ракурсах камер.

1.2 Нейросетевые подходы к решению задач обнаружения объектов

1.2.1 Семейство архитектур на основе R-CNN

R-CNN (Region-based Convolutional Neural Network) – архитектура (рисунок 1.4), разработанная командой из UC Berkeley для применения сверточных нейронных сетей (CNN) к задаче распознавания объектов [10].

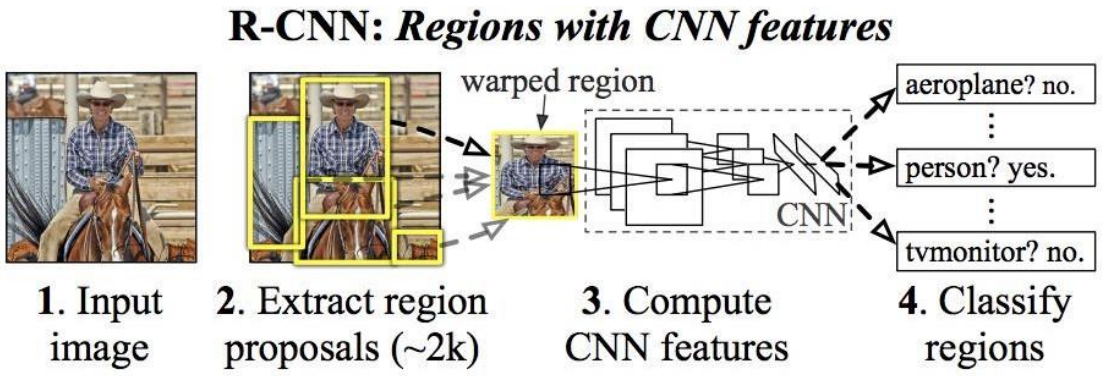

Рисунок 1.4 – Схема работы архитектуры R-CNN

Для решения данной задачи на вход CNN подается не все изображение целиком, а предварительно выделенные 2000 регионов, на которых предположительно имеются какие-то объекты. Выбор таких регионов основан на алгоритме выборочного поиска. Регионы-кандидаты деформируются в квадрат и передаются в сверточную нейронную сеть, которая выдает в качестве выходного вектора 4096-мерный вектор признаков. В качестве CNN-сети используется готовая архитектура – CaffeNet (AlexNet) [2]. Для классификации объектов извлеченные элементы передаются в SVM (Support Vector Machine), где последний классификационный слой CaffeNet был заменён на слой с N+1 выходами (с дополнительным классом для фона). Несмотря на то, что R-CNN неплохо показала себя в задаче семантического сегментирования [14] она еще обладала рядом проблем:

– обучение сети занимает огромное количество времени;

– для обработки одного изображения требуется около 47 секунд, поэтому данная архитектура не может быть реализована в режиме реального времени;

– алгоритм выборочного поиска может привести к генерации плохих предложений по регионам-кандидатам.

Поэтому в дальнейшем была разработана целая серия улучшений данной архитектуры:

1) fast R-CNN. Поскольку производительность R-CNN была невысока авторы реализовали две модификации, позволивших ускорить обработку тестовых изображений:

– пропускать через CNN не каждый из 2000 регионов-кандидатов по отдельности, а все изображение целиком. Таким образом, операция свертки выполняется только один раз для каждого изображения, и из него генерируется карта объектов;

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

Итоговая архитектура (рисунок 1.5) имеет значительно лучшие, чем у R-CNN результаты производительности в обучающих и тестовых сессиях.

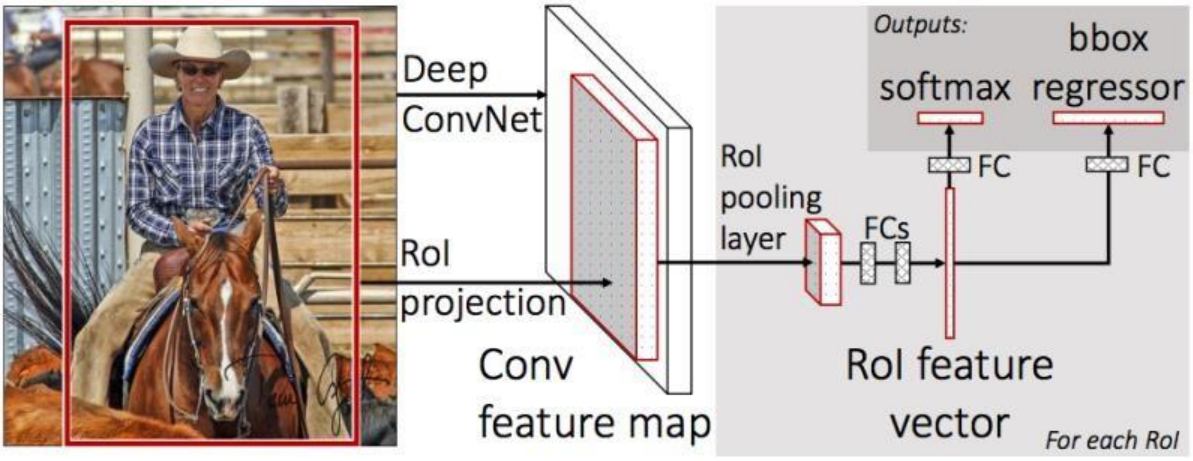

Рисунок 1.5 – Схема работы архитектуры Fast R-CNN

2) faster R-CNN добивается большего прогресса, чем Fast R-CNN [10]. Выборочный процесс поиска заменяется региональной сетью предложений (RPN). RPN - это сеть, предлагающая регионы. Подобно Fast R-CNN, изображение предоставляется в качестве входа в сверточную сеть, которая предоставляет сверточную карту признаков. Далее для прогнозирования предложений по регионам используется RPN. Предсказанные области предложений затем изменяются с использованием слоя пула RoI, который затем используется для классификации изображения в пределах предлагаемой области.

На рисунке 1.6 проиллюстрирована скорость обработки одного изображения каждой из архитектур семейства R-CNN.

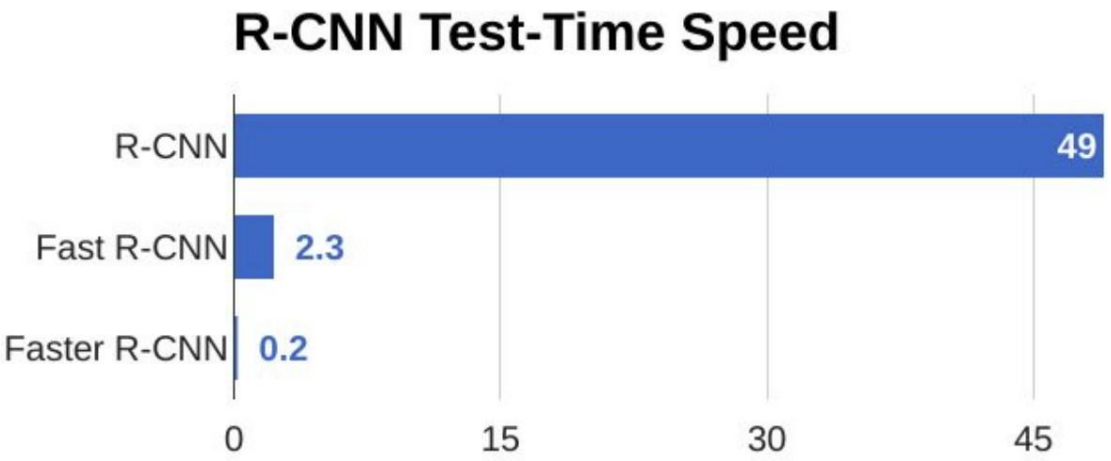

Рисунок 1.6 – Сравнение производительности R-CNN архитектур

1.2.2 Архитектура SSD

Архитектура SSD (Single Shot MultiBox Detector) (рисунок 1.7) использует единую нейронную сеть, которая выполняет все необходимые вычисления и устраняет необходимость в ресурсоёмких методах предыдущего поколения.

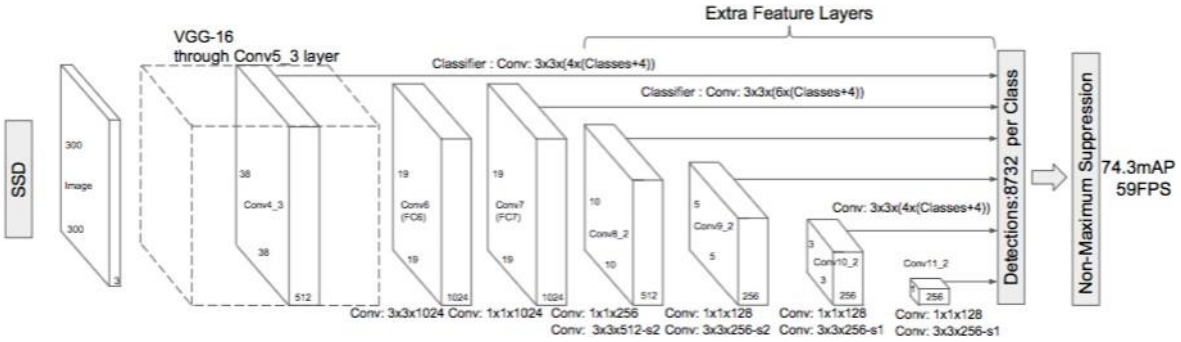

Рисунок 1.7 – Архитектура нейронной сети SSD

SSD основана на архитектуре VGG-16 (рисунок 1.8). Причиной использования VGG-16 в качестве базовой сети заключается в его высокой производительности в задачах классификации изображений высокого качества и его популярности для задач, где трансферное обучение помогает в улучшении результатов. Вместо оригинальных полностью связанных VGG слоев, набор вспомогательных сверточных слоев, что позволяет извлекать объекты в не-

скольких масштабах и постепенно уменьшать размер входных данных для каждого последующего слоя.

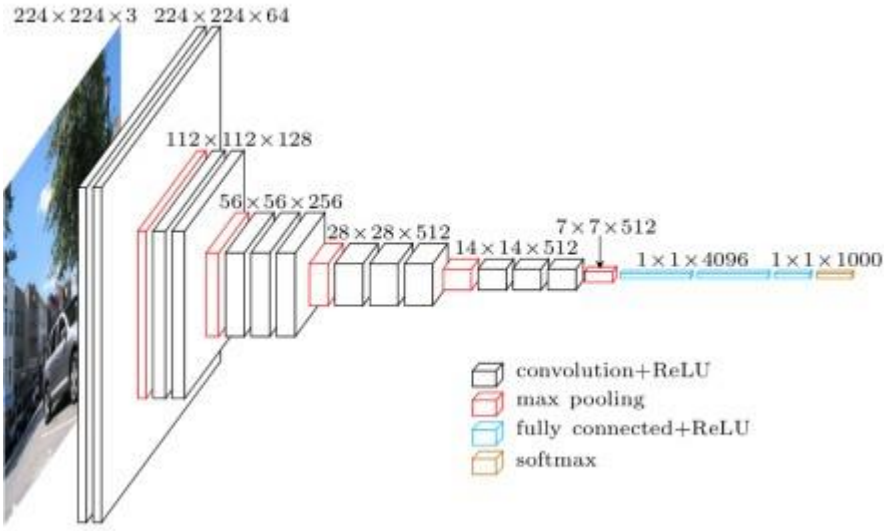

Рисунок 1.8 – Архитектура VGG

Данная нейронная сеть достигает точности равной 75,1% mAP, но она становится значительно хуже при обнаружении маленьких объектов так как они могут отображаться не на всех картах объектов.

1.2.3 Семейство архитектур на основе YOLO

Нейронные сети, построенные на базе архитектуры YOLO отличается от вышеописанных методов тем, что рассматривает проблему обнаружения изображений как проблему регрессии, а не как проблему классификации, и поддерживает одну сверточную нейронную сеть для выполнения всех задач. YOLOv1 (рисунок 1.9) реализован в виде сверточной нейронной сети и был оценен в наборе данных обнаружения PASCAL VOC. Он состоит из 24 сверточных слоев, за которыми следуют 2 полностью связанных слоя. Слои разделены по их функциональности следующим образом:

1) первые 20 сверточных слоев, за которыми следует средний пул и полностью подключенный слой, проходят предварительную подготовку по набору классификационных данных класса ImageNet1000;

2) предварительная подготовка к классификации проводится на базе данных с разрешением 224 x 224;

3) слои нейронной сети состоят из восстановительных слоев 1x1 и сверточных слоев 3x3;

4) последние 4 сверточных слоя, за которыми следуют 2 полностью связанных слоя, добавляются для обучения сети обнаружению объектов;

5) обнаружение объекта требует более детальной детализации, поэтому разрешение набора данных увеличено до 448 x 448;

6) последний слой предсказывает вероятности класса и ограничивающие рамки.

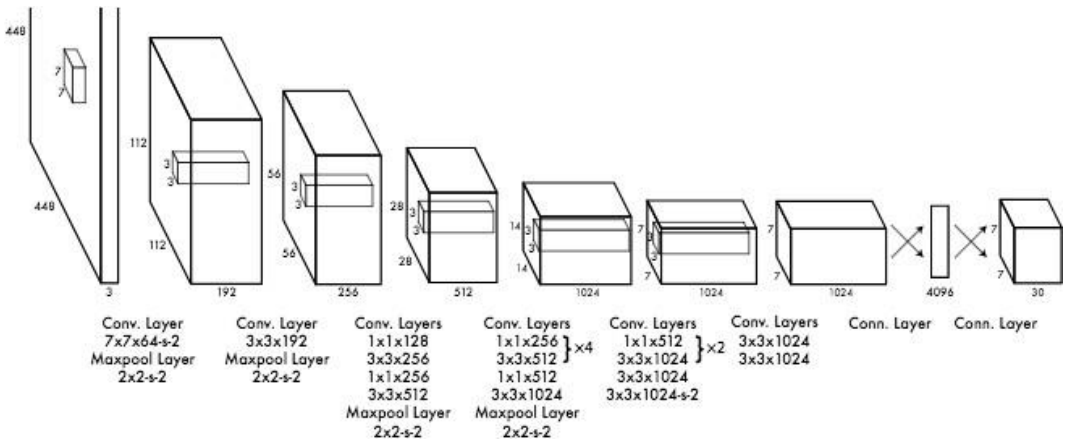

Рисунок 1.9 – Архитектура YOLO v1

В качестве входного изображения используется изображение 448 x 448, а выходным значением является предсказание класса объекта, заключенного в ограничивающий прямоугольник.

Объединение всех независимых задач в одну нейронную сеть дает архитектуре YOLO следующие преимущества:

1) высокая скорость обработки: YOLO чрезвычайно быстро сопоставляется со своими предшественниками, поскольку использует единую сеть свертки для обнаружения объектов. Свертка выполняется для всего входного изображения только один раз, чтобы получить предсказания;

2) сниженная вероятность появления ошибки фона: YOLO выполняет свертку всего изображения, а не его частей, благодаря чему кодирует контекстную информацию о классах и их появлении. Он делает меньше ошибок

при прогнозировании фоновых исправлений как объектов, поскольку просматривает все изображение и признаки глобально, а не локально;

3) обобщающая способность: YOLO изучает обобщенные представления объектов, благодаря которым он может быть применен к новым представлениям объектов.

Данные преимущества позволяют существенно улучшить производительность этой архитектуры по сравнению с остальными (рисунок 1.10), при этом существенно не проигрывая в точности.

Модель	mAP	Время
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

Рисунок 1.10 – Сравнение точности и производительности архитектур сверточных нейронных сетей

Впоследствии данная архитектура улучшалась, в результате чего были выпущены улучшения – YOLOv2 и YOLOv3 (рисунок 1.11).

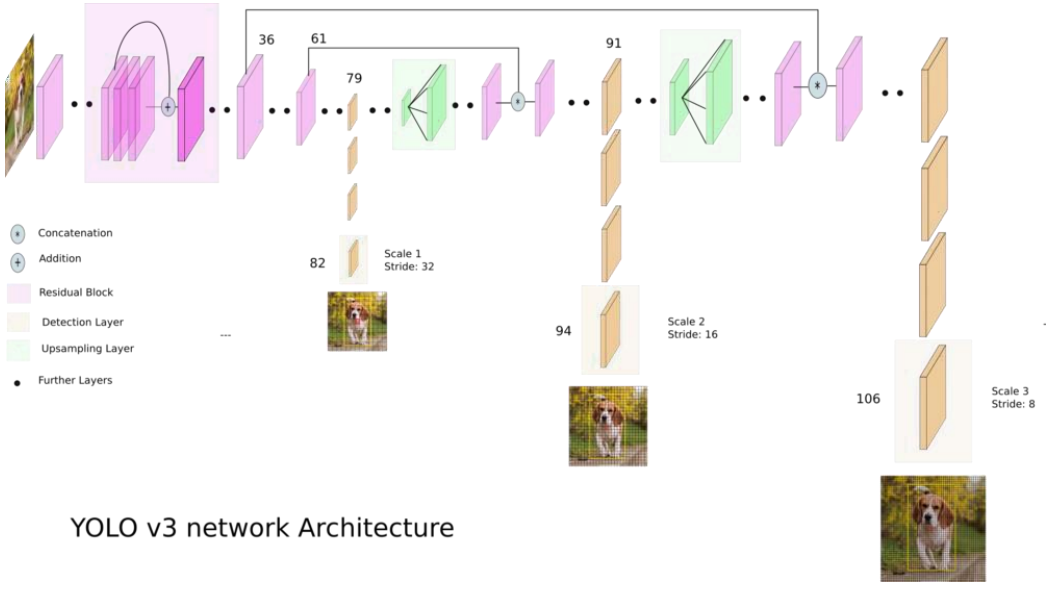

Рисунок 1.11– Архитектура YOLOv3

Главная особенность новой версии состоит в том, что на выходе есть три слоя каждый из которых рассчитан на обнаружения объектов разного размера.

1.3 Обзор библиотек для работы с нейронными сетями

1.3.1 Библиотека PyTorch

PyTorch – библиотека машинного обучения для языка Python с открытым исходным кодом, разрабатываемая группой искусственного интеллекта Facebook. Используется для решения различных задач, таких, как компьютерное зрение и обработка естественных языков [3].

PyTorch предоставляет две основные высокоуровневые модели:

– тензорные вычисления с развитой поддержкой ускорения на графическом ускорителе;

– глубокие нейронные сети на базе системы autodiff.

1.3.2 Библиотека TensorFlow

TensorFlow – это библиотека, созданная компанией Google, используемая при разработке систем, включающих технологии машинного обучения. Данная библиотека включает в себя реализацию множества сложных алгоритмов, рассчитанных на решение задач машинного обучения [1].

Среди достоинств данной библиотеки можно выделить следующие:

– большое количество руководств и документации;

– мощные средства мониторинга процесса обучения моделей и визуализации (Tensorboard);

– поддержка библиотеки большим сообществом разработчиков и техническими компаниями;

– поддержка распределенного обучения;

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

1.4 Выводы

Существующие решения в области обнаружения объектов имеют существенные различия, а также свой ряд достоинств и недостатков, что позволяет делать выбор конкретного решения исходя из поставленной задачи. В данной работе важной особенностью поставленной задачи является обработка видеопотока в режиме реального времени. Исходя из этого целесообразно остановиться на архитектуре нейронной сети с высокой производительностью. Нейронная сеть YOLO имеет преимущества, важные для задачи, поставленной в данной работе. Одним из определяющих достоинств является скорость обработки видеокадров при небольшой потере в точности (рисунок 1.9), поэтому в качестве используемой в данной работе архитектуры была выбрана именно она.

2 АРХИТЕКТУРА И ОБУЧЕНИЕ НЕЙРОННОЙ СЕТИ YOLOV3

2.1 Постановка задачи

Пусть $X = \{X_i\}_1^n$ – множество матриц (набор некоторых изображений), $Y = \{Y_i\}_{i=1}^n$ – множество элементов (набор пар координат, соответствующих верхнему левому и нижнему правому углам объекта) таких, что:

$$Y_i = \{y_{ij}\}_{j=1}^m \, y_{ij} = \{t = (t_1; t_2): t_k \in R^2\}, \quad (2.1)$$

f – целевая функция, отображающая множество X на множество Y:

$$f: X \to Y. \quad (2.2)$$

Пусть $X^* \subset X$ – набор изображений обучающей выборки, тогда значения целевой функции известны на наборе пар $(X_i^*; Y_i^*)$:

$$Y^* = f(X^*), \quad (2.3)$$

Необходимо построить алгоритм A, позволяющий восстановить функциональную зависимость между изображениями и координатами объектов на изображениях:

$$A: X^* \to Y^*. \quad (2.4)$$

Алгоритм A должен обладать следующими свойствами:

1) A должен приближать целевую функцию не только на множестве объектов обучающей выборки, но и на всем множестве X;

2) A должен допускать численную реализацию.

2.2 Исходные данные

В качестве исходных данных были использованы кадры с 40 видеопотоков уличных камер видеонаблюдения в г. Челябинск (рисунок 2.1). Всего было получено около 10 тысяч изображений с разрешением 1980x1080 пикселей.

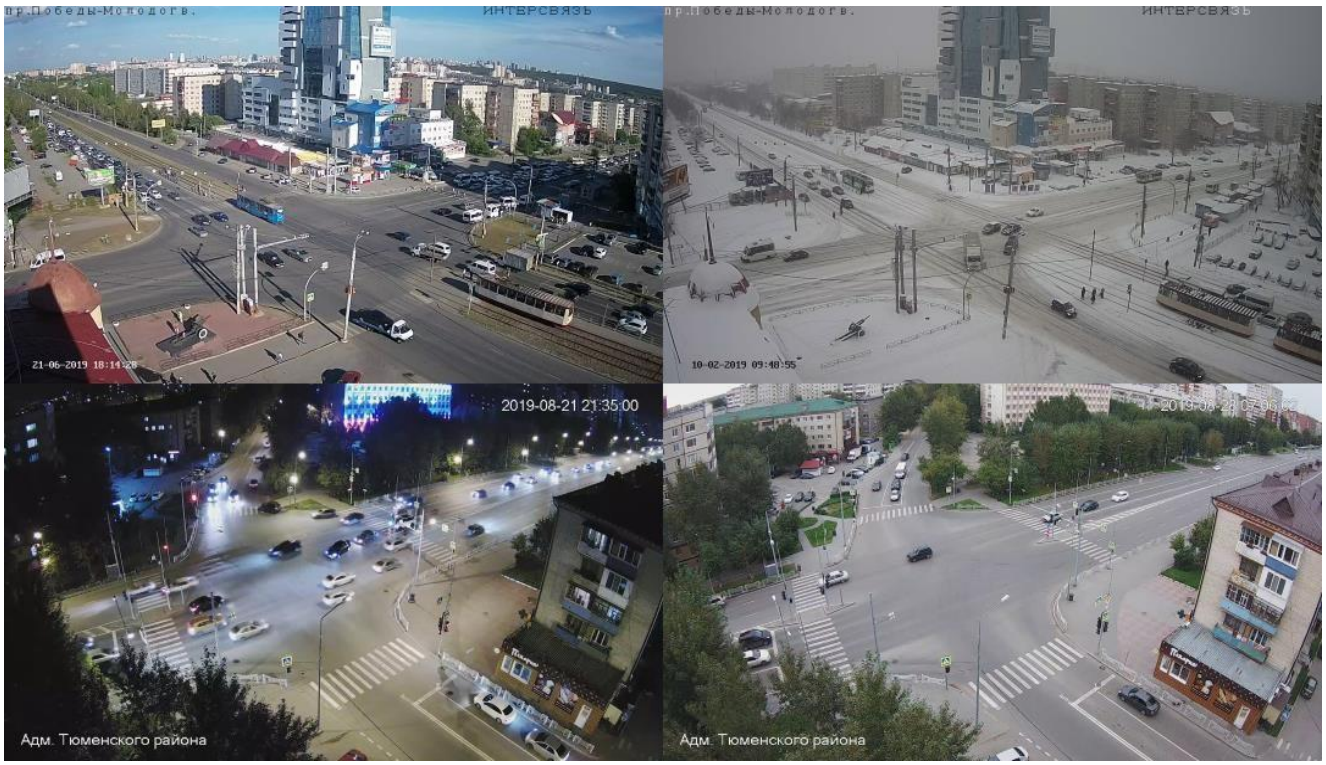

Рисунок 2.1 – Примеры исходных изображений

Транспортные средства в исходных данных были разделены на следующие классы:

– легковые автомобили;

– малые и особо малые автобусы;

– средние автобусы;

– большие автобусы;

– малые грузовики (грузоподъемностью до 2-х тонн);

– средние грузовики (с грузоподъемностью от 2-х до 6 тонн);

– большие грузовики (с грузоподъемностью от 6 тонн);

– троллейбусы;

– специальный транспорт (тракторы, бетономешалки и т. д.);

– трамваи;

– автопоезда.

Примеры изображений данных классов можно увидеть на рисунке 2.2.

Рисунок 2.2 – Примеры изображений классов транспортных средств

2.3 Архитектура нейронной сети

Архитектура нейронной сети YOLOv3 (рисунок 2.3) состоит из 106 слоев и является модификацией нейронной сети Darknet-53 (рисунок 2.4), включающей в себя 53 слоя и обученной на наборе данных ImageNet. Кроме этого, она включает в себя еще 53 слоя с двумя выходными слоями различной размерности, что позволяет проводить обнаружение в трех различных масштабах. Данная модификация способствует более точному распознаванию объектов различного масштаба.

В качестве входных данных YOLOv3 принимает изображение, представимое в виде трехмерного тензора размером $h \times w \times 3$, где h, w – высота и длина входного изображения. Размерность выходных слоев определяется уменьшением размера входного изображения в 32, 16, 8 раз, соответственно.

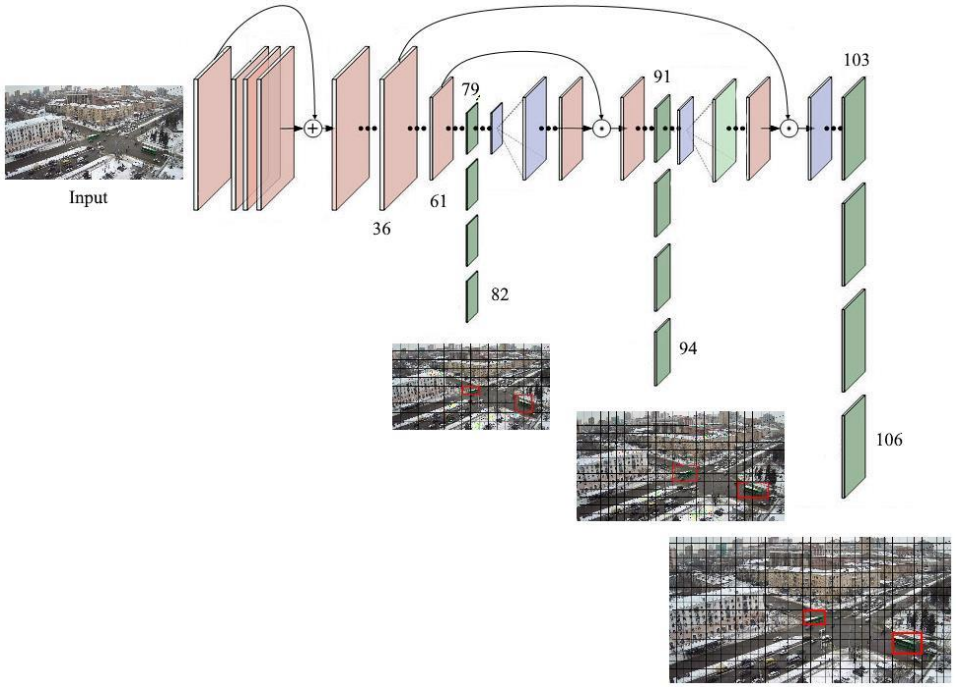

Рисунок 2.3 – Архитектура нейронной сети YOLOv3

	Type	Filters	Size	Output
	Convolutional	32	3 x 3	256 x 256
	Convolutional	64	3 x 3 / 2	128 x 128
1 x	Convolutional	32	1 x 1	
	Convolutional	64	3 x 3	
	Residual			128 x 128
	Convolutional	128	3 x 3 / 2	64 x 64
2 x	Convolutional	64	1 x 1	
	Convolutional	128	3 x 3	
	Residual			64 x 64
	Convolutional	256	3 x 3 / 2	32 x 32
8 x	Convolutional	128	1 x 1	
	Convolutional	256	3 x 3	
	Residual			32 x 32
	Convolutional	512	3 x 3 / 2	16 x 16
8 x	Convolutional	256	1 x 1	
	Convolutional	512	3 x 3	
	Residual			16 x 16
	Convolutional	1024	3 x 3 / 2	8 x 8
4 x	Convolutional	512	1 x 1	
	Convolutional	1024	3 x 3	
	Residual			8 x 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Рисунок 2.4 – Архитектура Draknet-53

Данная архитектура включает в себя следующие виды слоев:

– сверточный слой;

– слой повышенной дискретизации;

– слой маршрутизации;

– выходной слой.

Сверточный слой представляет из себя набор карт, представляющих собой матрицы чисел, количество которых зависит от уровня слоя [21]. У каждой карты имеется синаптическое ядро, представляющее собой матрицу весов. Синаптическое ядро необходимо для вычисления сверточной карты текущего слоя (рисунок 2.5) путем преобразования карты предыдущего слоя методом скользящего окна:

$$A' = \{a_{ij}\}_{i=1,j=1}^{n,m},\ a_{ij} = \sum_{k=0}^{h-1}\sum_{l=0}^{w-1} A_{n+k,m+l} * C_{k,l}, \qquad (2.5)$$

где A' – карта предыдущего слоя,

C – синаптическое ядро,

A – карта текущего слоя,

h, w – размерность синаптического ядра.

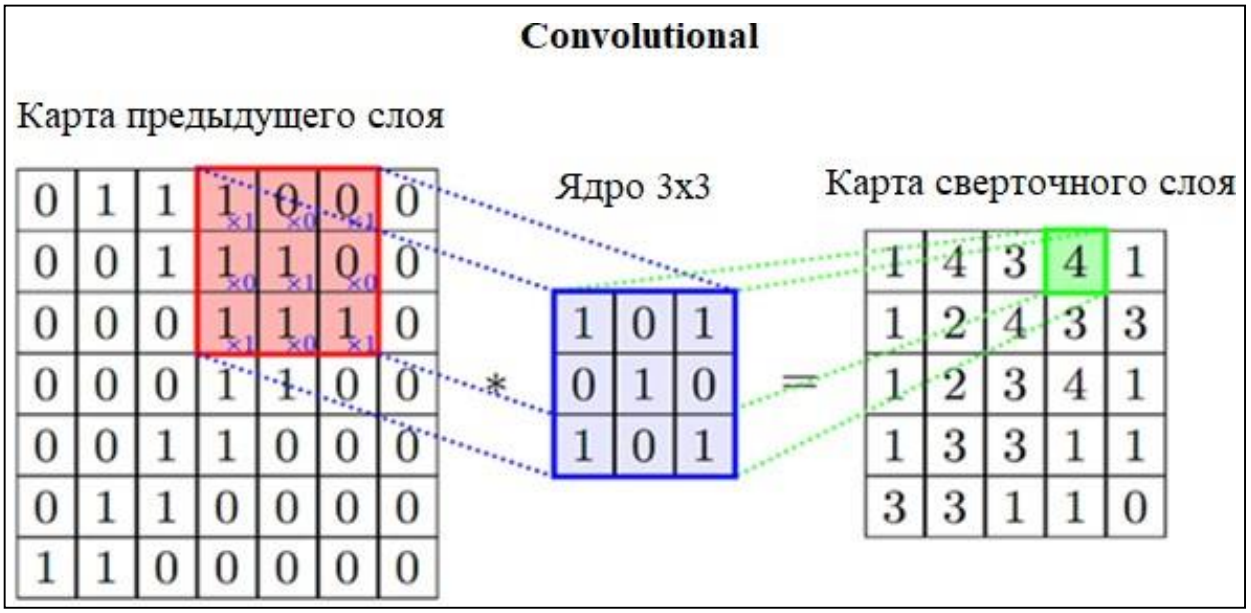

Рисунок 2.5– Операция свертки и получение значений сверточной карты

Слой повышенной дискретизации необходим для увеличения размерности карты, поступающей с предыдущего слоя. В архитектуре нейронной сети YOLOv3 содержится два слоя повышенной дискретизации, которые увеличивают размерность в два раза. На рисунке 2.6 проиллюстрирована логика работы данного слоя.

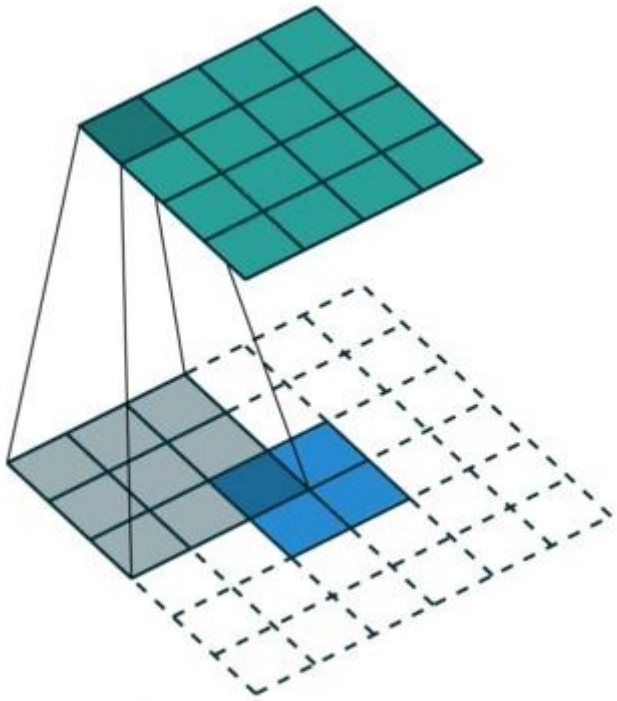

Рисунок 2.6 – Иллюстрация работы слоя повышенной дискретизации

Данный слой работает по принципу скользящего окна [22]. Пусть на входе данному слою поступает карта, размерности $\mathrm{n} \times \mathrm{m}$. Тогда размер окна будет $(\mathrm{n}+1) \times (\mathrm{m}+1)$. Карта, полученная с предыдущего слоя увеличивается до размерности $3\mathrm{n} \times 3\mathrm{m}$, при этом исходная карта помещается в центр, а остальные элементы становятся равны 0 (на рисунке 2.6 соответствующие ячейки изображены пунктиром). После этого выполняется операция, тождественная операции свертки, приведенной в формуле 2.5, с использованием синаптического ядра, в котором все элементы $C_{k,l}$ равны 1.

Слой маршрутизации необходим для объединения карт различных слоев одинаковой размерности [24]. Пусть Route_{nm} – некоторый слой маршрутизации, тогда множество карт, полученное на выходе данного слоя можно определить следующим образом:

$$L^R_{nm} = L_n \cap L_m, \tag{2.6}$$

где L^R_{nm} – множество карт, полученное на выходе слоя маршрутизации Route_{nm};

L_n – множество карт, полученное на выходе слоя n;

L_m – множество карт, полученное на выходе слоя m.

Выходной слой содержит данные об обнаруженных объектах. Для обнаружения объектов, которые имеют не равные размеры длины и высоты ограничивающего прямоугольника используются опорные прямоугольники (рисунок 2.7).

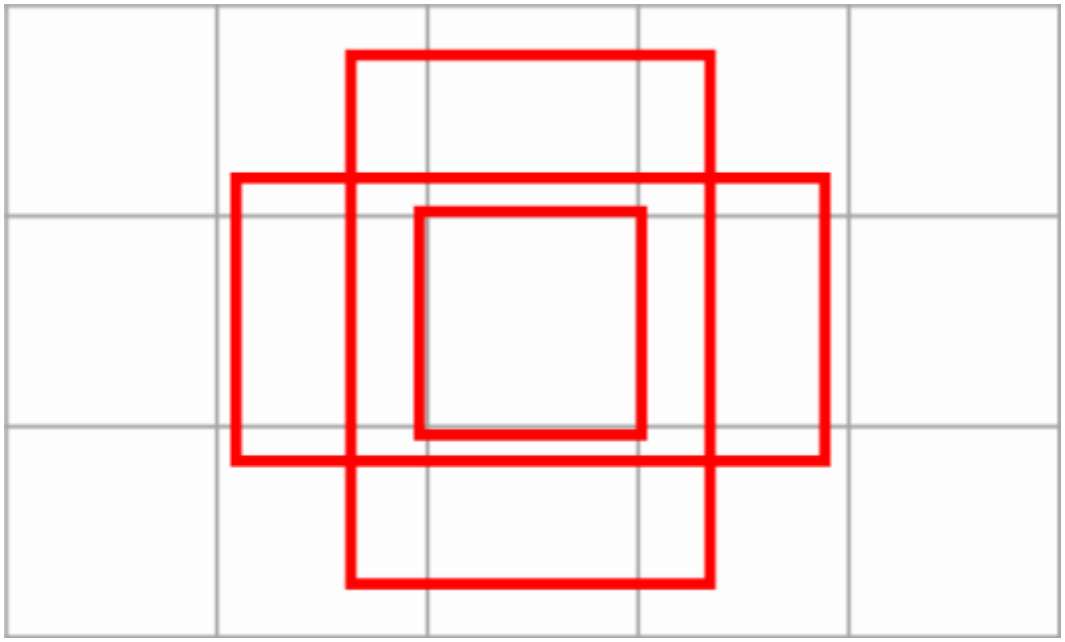

Рисунок 2.7 – Опорные прямоугольники для одной ячейки

Количество и размеры данных прямоугольников должны быть заданы отдельно для каждого выходного слоя. Выходной слой представляет собой трехмерную матрицу, длина и высота которой определяется размерами карты, полученной от предыдущего слоя. Размерность выходного слоя в глубину определяется следующим выражением:

$$S = B(5 + C), \tag{2.7}$$

где S – размерность выходного слоя в глубину;

B – количество опорных прямоугольников;

C – количество классов;

Число 5 в формуле 2.7 определяет пять чисел для каждой ячейки выходного слоя, в которых содержится информация о координатах, размерах ограничивающего прямоугольника и вероятности нахождения в нем объекта.

2.4 Подготовка данных

Для обучения нейронной сети исходные данные были размечены. В результате было получено около 430 000 объектов транспортных средств различных классов. Индексация классов и соответствующие им цвета, используемые в дальнейшем для отображения результатов обнаружения, представлены в таблице 2.1.

Таблица 2.1 – Индексация и цвета классов

Индекс	Класс	Цвет
0	Легковой автомобиль	Желтый
1	Малый автобус	Синий
2	Средний автобус	Оранжевый
3	Большой автобус	Зеленый
4	Малый грузовик	Красный
5	Средний грузовик	Коричневый
6	Большой грузовик	Черный
7	Троллейбус	Фиолетовый
8	Спец. транспорт	Серый
9	Трамвай	Розовый
10	Автопоезд	Голубой

Входные данные представлены следующим образом: изображение в формате JPG или PNG (рисунок 2.8) и текстовый файл с разметкой в следующем формате:

$$\begin{matrix} C_1 & X_1 & Y_1 & W_1 & H_1 \\ C_2 & X_2 & Y_2 & W_2 & H_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ C_i & X_i & Y_i & W_i & H_i \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ C_n & X_n & Y_n & W_n & H_n, \end{matrix} \quad (2.8)$$

где i – номер объекта;

n – количество объектов на изображении;

C_i – индекс класса -ого объекта;

X_iY_i – координаты центра прямоугольника, содержащего объект;

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

Параметры $X_iY_iW_iH_i$ записаны в относительных от размера изображения величинах ($X_iY_iW_iH_i \in [0; 1]$).

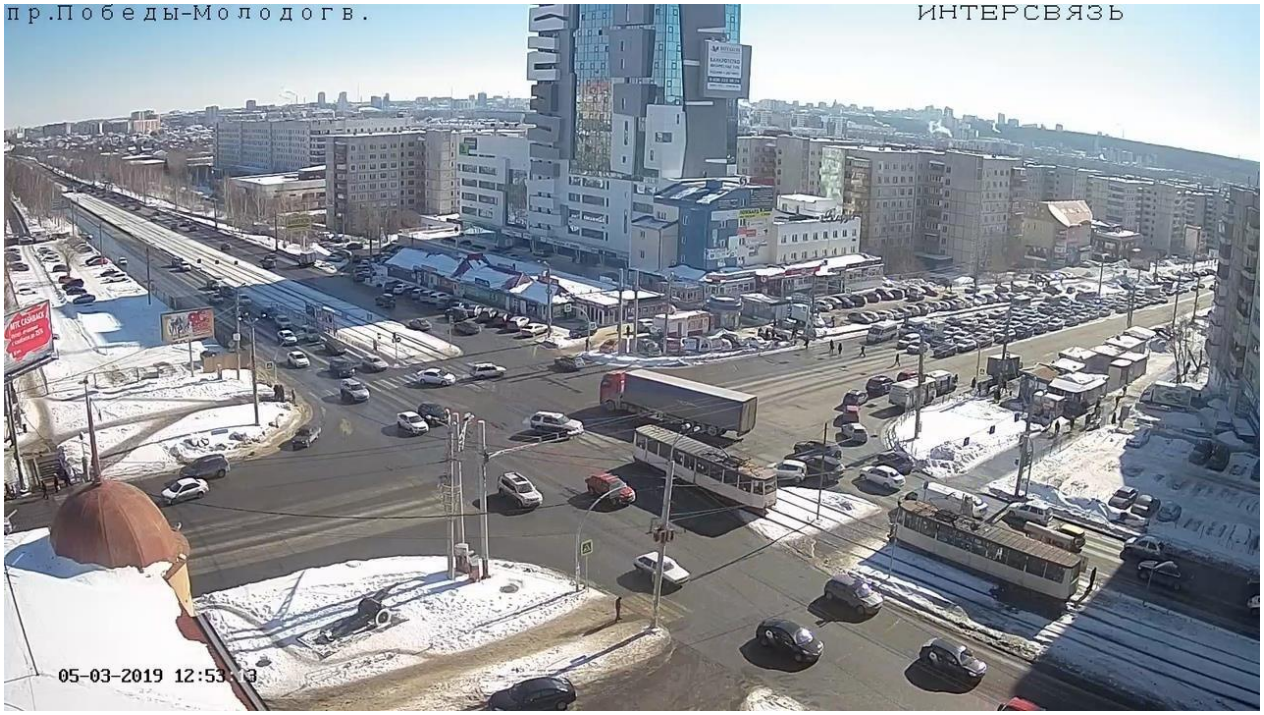

Рисунок 2.8 – Пример изображения обучающей выборки

Для более качественного обучения нейронной сети набор данных был расширен с помощью аугментации, что увеличило набор данных в 10 раз [25]. Для аугментации были применены следующие преобразования в различных комбинациях: горизонтальное отображение; аффинное и перспективное преобразования; наложение шума; искажение цветов (рисунок 2.9).

Итоговый набор данных составил 4.3 млн. объектов. Распределение объектов каждого класса в обучающей выборке представлено в таблице 2.2. Полученные в результате аугментации данные были разбиты на обучающую и тестовую выборки в соотношении 80/20%.

Рисунок 2.9 – Пример аугментированных данных

Таблица 2.2 – Распределение объектов в обучающей выборке

Класс	Количество объектов	Отношение к общему количеству
Легковой автомобиль	3518370	74,5%
Малый автобус	228810	4,8%
Средний автобус	145680	3,1%
Автобус	163430	3,5%
Малый грузовик	184520	3,9%
Средний грузовик	112240	2,4%
Грузовик	95230	2,0%
Троллейбус	112690	2,4%
Спец. Транспорт	38620	0,8%
Трамвай	91540	1,9%
Автопоезд	29560	0,6%

2.5 Обучение нейронной сети YOLOv3

Для обучения нейронной сети YOLOv3 используется метод обратного распространения ошибки с использованием градиентного спуска [13]. В основе данного метода лежит использование выходной ошибки нейронной сети для вычисления величин коррекции весов нейронов в ее скрытых слоях. Алгоритм является итеративным и использует принцип обучения «по эпохам», при котором изменение весов производится после подачи на вход нейронной сети нескольких экземпляров обучающего множества, а ошибка усредняется по всем примерам:

$$\mathrm{E} = \frac{1}{n}\sum_{i=1}^{n} loss_i, \tag{2.9}$$

где n – количество объектов в одной итерации обучения;

$loss_i$ – значение функции потерь для -го объекта выборки;

E – усредненная ошибка на одной итерации.

На каждой итерации происходит несколько прямых проходов нейронной сети и один обратный. На прямом входной вектор распространяется от входов сети к ее выходам и формирует некоторый выходной вектор, соответствующий текущему состоянию весов. Затем вычисляется ошибка нейронной сети, как разность между фактическим и целевым значениями. На обратном проходе эта ошибка распространяется от выхода сети к ее входам [7], и производится коррекция весов нейронов в соответствии с формулой:

$$\Delta w_{ij} = -\mu \frac{\partial E}{\partial w_{ij}}, \tag{2.10}$$

где w_{ij} – вес -ой связи j-того нейрона;

μ – параметр скорости обучения;

Данный способ обучения, по сравнению с обучением «по шагам», является более устойчивым к выбросам и аномальным значениям целевой переменной за счет усреднения ошибки по многим примерам.

Общий алгоритм обратного распространения ошибки можно описать следующим образом:

1) инициализировать веса, количество итераций I;
2) повторять для каждой итерации:

а) прямой ход для каждого слоя:

$$y_i = f_i\Big(\sum_{j \in Output} w_{ji}x_j + w_{0i}\Big), \tag{2.11}$$

где w_{ji} – весовой коэффициент связи -го и i-го нейронов;

x_j – сигнал с -го нейрона предыдущего слоя;

w_{0_i} – сигнал нейрона -го смещения текущего слоя;

$Output$ – множество нейронов предыдущего слоя;

f_{ki} – функция активации -го нейрона текущего слоя;

б) вычисление ошибки выходного слоя:

$$\delta_{Mi} = E, \tag{2.12}$$

где $loss$ – усредненная функция потерь;

δ_{Mi} – ошибка сигнала с -го нейрона выходного слоя M;

в) обратный ход:

$$\delta_{ki} = \sum_{j \in Output_{k+1}} w_{ji}\delta_{k+1j}, \tag{2.13}$$

где δ_{k+1j} – ошибка сигнала -го нейрона слоя $k+1$;

δ_{kj} – ошибка сигнала -го нейрона слоя k;

w_{ji} – весовой коэффициент связи -го и i-го нейронов;

$Output_{k+1}$ – множество нейронов слоя $k+1$;

г) градиентный шаг для каждого слоя:

$$w_{ji} = w_{ji} + \mu\delta_i f_i'(y_i), \tag{2.14}$$

где w_{ji} – весовой коэффициент связи -го и i-го нейронов;

μ – скорость обучения;

δ_i – ошибка -го нейрона предыдущего слоя;

y_i – сигнал -го нейрона предыдущего слоя.

2.5.1 Функции активации

Функция активация в алгоритме обратного распространения ошибки должна обладать несколькими важными характеристиками: непрерывностью, дифференцируемостью и, кроме этого, она должна являться монотонно неубывающей. Также для ради эффективности вычислений, необходимо, чтобы нахождение производной функции активации не было трудоемким. С учетом необходимых характеристик, в качестве функции активации была выбрана функция leaky ReLU, изображенная на рисунке 2.10. Данная функция вычисляется следующим образом:

$$f_{leakyReLU}(x) = \begin{cases} x, \text{если } x \geq 0, \\ 0.1x, \text{если } x < 0, \end{cases} \quad (2.15)$$

где $f_{leakyReLU}$ – функция Leaky ReLU.

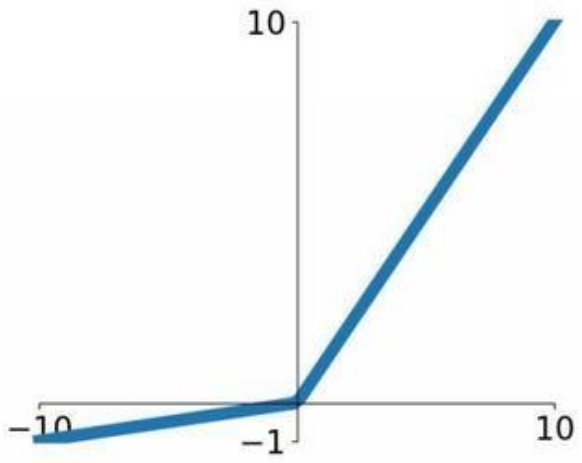

Рисунок 2.10 – Функция активации leaky ReLU

В качестве основных достоинств использования данной функции в качестве функции активации можно выделить следующие:

– отсутствие затухания градиента в процессе обучения;

– функция ReLU лишена трудоемких операций, что способствует тратить меньше времени на обучение.

2.5.2 Функция потерь

Для расчета потерь используется сумма квадратов ошибок между прогнозами и истинной величиной. Функция потерь состоит из трех компонен-

тов, отвечающих за точность различных показателей выходных данных нейронной сети:

– функция потерь для точности классификации;

– функция потерь для точности локализации;

– функция потерь для точности показателя достоверности.

Функция потерь для классификации показывает насколько точно нейронная сеть классифицировала определенный объект. Если объект обнаружен, потеря классификации в каждой ячейке является квадратичной ошибкой условных вероятностей для каждого класса:

$$E_c = \sum_{i=0}^{S^2} 1_i^{obj} \sum_{c \in classes} (p_i(c) - \tilde{p}_i(c))^2, \quad (2.16)$$

где 1_{ij}^{obj} – параметр, значение которого равно 1 если объект находится в ячейке i, иначе значение равно 0;

S^2 – количество ячеек, на которые было разбито изображение;

$p_i(c)$ – вычисленная вероятность принадлежности объекта классу c в ячейке i;

$\tilde{p}_i(c)$ – условная вероятность принадлежности объекта классу c в ячейке i;

E_c – функция потерь для классификации.

Условная вероятность определяется исходя из обучающего набора данных. Для класса, определенного для этого объекта в обучающей выборке, условная вероятность равна 1, а для других – 0.

Функция потерь для локализации показывает насколько точно нейронная сеть определила координаты и положение прямоугольника, ограничивающего объект. Данная функция потерь вычисляется следующим образом:

$$E_l = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} ((x_i - \tilde{x}_i)^2 + (y_i - \tilde{y}_i)^2) + \\ + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} ((\sqrt{w_i} - \sqrt{\tilde{w}_i})^2 + (\sqrt{h_i} - \sqrt{\tilde{h}_i})^2), \quad (2.17)$$

где 1_{ij}^{obj} – параметр, значение которого равно 1 если объект j находится в ячейке i, иначе значение равно 0;

B – количество найденных объектов;

S^2 – количество ячеек, на которые было разбито изображение;

x_i, y_i – вычисленные координаты ограничивающего прямоугольника;

$\tilde{x}, \tilde{y}$ – условные координаты ограничивающего прямоугольника;

w_i, h_i – вычисленные размеры ограничивающего прямоугольника;

$\tilde{w}, \tilde{h}$ – условные размеры ограничивающего прямоугольника;

E_l – функция потерь для локализации.

Условные координаты и размеры ограничивающего прямоугольника определяются из набора данных для каждого объекта.

Функция потерь для показателя достоверности показывает насколько точно нейронная сеть определяет вероятность нахождения объекта в ячейке. Данная функция состоит из двух частей. Если в ячейке обнаружен объект, потеря доверия определяется следующей формулой:

$$E_p^{obj} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} (C_i - \tilde{C}_i)^2, \quad (2.18)$$

где 1_{ij}^{obj} – параметр, значение которого равно 1 если объект j находится в ячейке i, иначе значение равно 0;

B – количество найденных объектов;

S^2 – количество ячеек, на которые было разбито изображение;

C_i – вычисленный показатель достоверности для ячейки i;

$\tilde{C}$ – условный показатель достоверности для ячейки i;

E_p^{obj} – функция потерь для показателя достоверности, если объект обнаружен.

Условный показатель достоверности в данном случае равен 1 если в обучающем наборе в соответствующей ячейке есть объект, 0 – если объекта нет. Если объект в ячейке не обнаружен, функция потерь для достоверности определяется следующей формулой:

$$E_p^{noobj} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{noobj} (C_i - \tilde{C}_i)^2, \quad (2.19)$$

где 1_{ij}^{noobj}– параметр, значение которого равно 0 если объект j находится в ячейке i, иначе значение равно 1;

B – количество найденных объектов;

S^2 – количество ячеек, на которые было разбито изображение;

C_i – вычисленный показатель достоверности для ячейки i;

$\tilde{C}_i$ – условный показатель достоверности ячейки i.

E_p^{noobj} – функция потерь для показателя достоверности, если объект не обнаружен.

Таким образом, функция потерь E_p для показателя достоверности равна сумме двух функций для разных случаев:

$$E_p = E_p^{obj} + E_p^{noobj}, \quad (2.20)$$

В качестве основной функции потерь, используется функция $loss$, которая является суммой трех, описанных выше, функций потерь:

$$loss = E_c + E_l + E_p, \quad (2.21)$$

где E_c– функция потерь для классификации;

E_l– функция потерь для локализации;

E_p– функция потерь для показателя достоверности.

2.6 Метрики качества

Для контроля переобучения и для сравнения различных моделей при обучении были использованы метрики качества, определяющие качество обученной модели.

Метрика Intersection over Union (IoU) (рисунок 2.11) показывает, насколько у двух объектов, эталонного и текущего, совпадает внутренняя площадь.

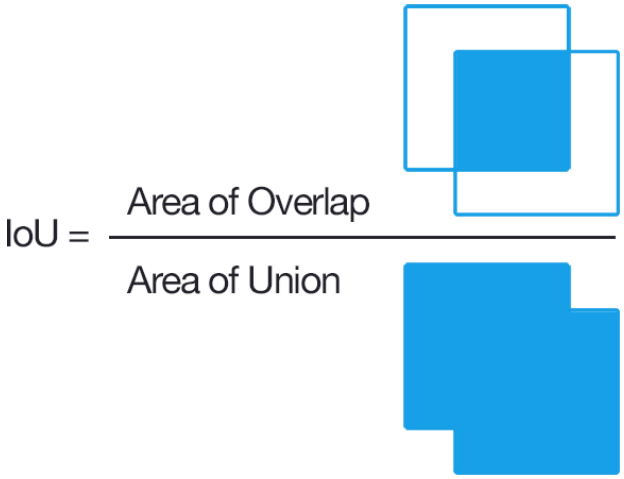

Рисунок 2.11 – Визуальное представление метрики IoU

Пусть A – множество пикселей, которые нейронная сеть отнесла к объекту, а B – множество пикселей, фактически принадлежащих объекту. Тогда значение метрики IoU можно описать следующим образом:

$$\mathrm{IoU} = \frac{|A \cap B|}{|A \cup B|}. \tag{2.22}$$

Для оценки качества работы алгоритма на каждом из классов по отдельности используются метрики precision и recall (рисунок 2.12). Precision можно интерпретировать как долю объектов, названных классификатором положительными и при этом действительно являющимися положительными. Recall показывает, какую долю объектов положительного класса из всех объектов положительного класса нашел алгоритм. Формально данные метрики можно описать следующими выражениями:

$$P = \frac{O \cap D}{D}, \tag{2.23}$$

$$R = \frac{O \cap D}{O}, \tag{2.24}$$

где O – доля пикселей, принадлежащих объекту;

D – множество пикселей, которые нейронная сеть отнесла к объекту;

P– метрика точности;

R– метрика полноты.

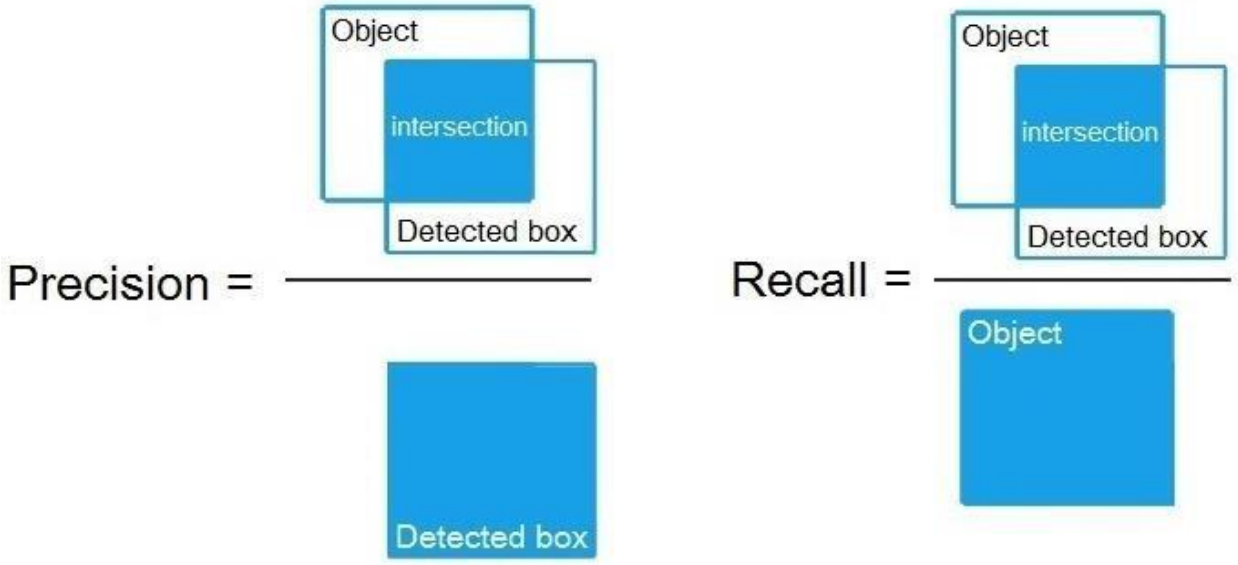

Рисунок 2.12 – Вычисление метрик precision и recall

Эти значения можно рассчитать на основании матрицы ошибок, которая составляется для каждого класса отдельно (рисунок 2.13). В таблице содержится информация сколько раз система приняла верное и сколько раз неверное решение по каждому классу:

– TP – истинно-положительное решение;

– TN – истинно-отрицательное решение;

– FP – ложноположительное решение;

– FN – ложноотрицательное решение.

		Истинный результат	
		Класс А	Не класс А
Предсказанный результат	Класс А	TP	FP
	Не класс А	FN	TN

Рисунок 2.13 – Матрица ошибок для некоторого класса А

Тогда, точность и полнота определяются следующим образом:

$$P = \frac{TP}{TP + FP}, \tag{2.25}$$

$$R = \frac{TP}{TP + FN}. \tag{2.26}$$

Используя матрицу ошибок, формулу для нахождения значения метрики IoU можно записать следующим образом:

$$IoU = \frac{TP}{TP + FP + FN}. \tag{2.27}$$

Для оценки общей точности распознавания по всем классам используется метрика mAP (mean average precision). Она является средним значением между значениями метрики AP (average precision) по всем классам. Для вычисления значения AP для определенного класса вычисляется площадь под кривой, построенной с использованием метрик точности и полноты для этого класса (рисунок 2.14):

$$AP = \int_0^1 p(r)dr, \tag{2.28}$$

где p – точность,

r – полнота,

AP – средняя точность по классу.

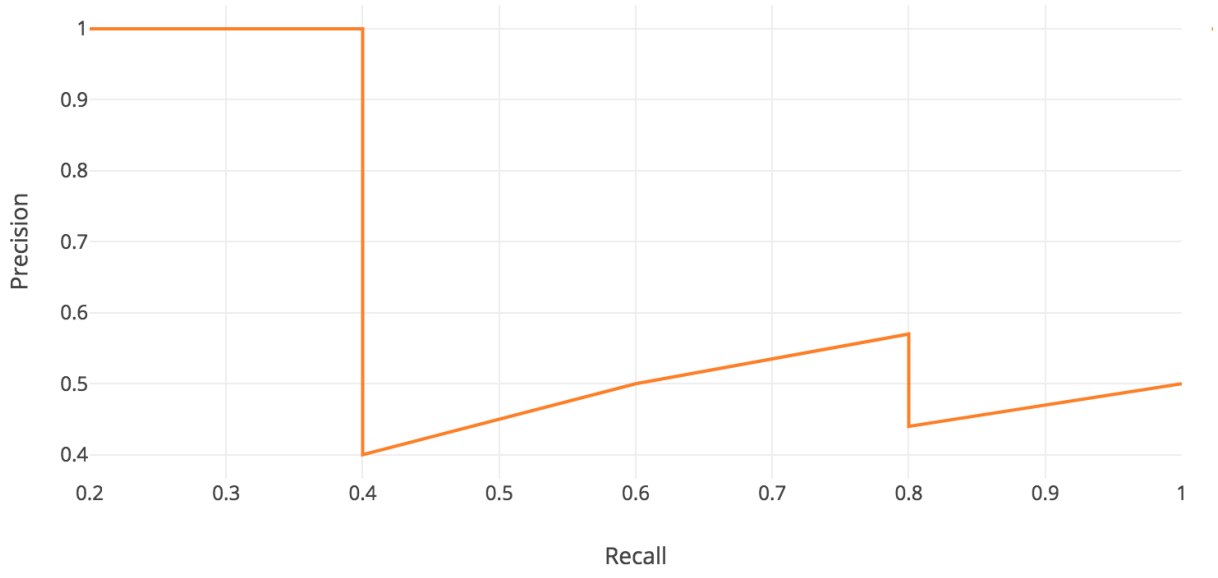

Рисунок 2.14 – Пример PR-кривой

После нахождения значений метрики AP для каждого класса, значение метрики mAP можно вычислить следующим образом:

$$mAP = \frac{1}{n} \sum_{i=1}^{n} AP_i, \tag{2.29}$$

где n – количество классов в выборке;

AP_i – среднее значение точности по классу i.

2.7 Подсчет транспортных средств

Для подсчета транспортных средств по различным направлениям введем следующее множество:

$$C = \{c_i\}_{i=1}^{m}, c_i \in N, \tag{2.30}$$

где C – множество чисел, определяющих количество транспортных средств в каждом направлении;

c_i – количество транспортных средств, проехавших в -ом направлении;

N – множество натуральных чисел;

m – количество направлений на дорожном узле.

Определим для каждого направления специальные отрезки, при пересечении которых значение соответствующего элемента в наборе C будет увеличиваться (рисунок 2.15).

Рисунок 2.15 – Пересечение
линии подсчета

Для решения данной задачи необходимо обнаруживать пересечение прямоугольника, ограничивающего объект, и отрезков каждого направления движения (рисунок 2.16).

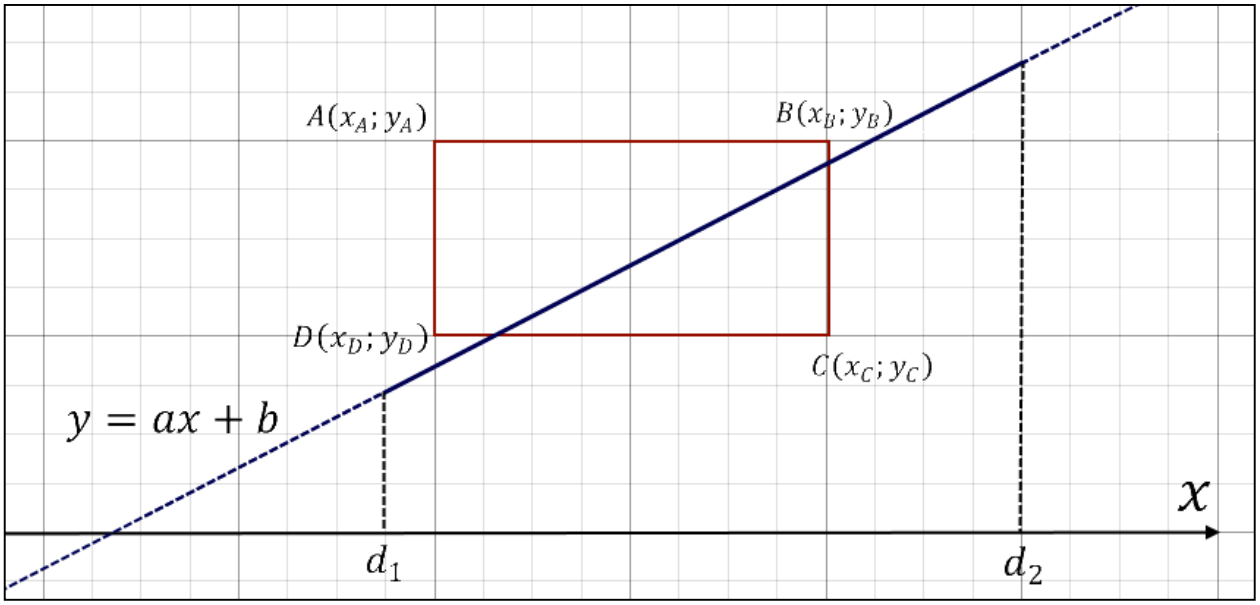

Рисунок 2.16 – Пересечение отрезка и ограничивающего прямоугольника

Пусть уравнение прямой, на котором лежит отрезок -го направления задается следующим образом:

$$y = a_i x + b_i, \tag{2.31}$$

где $a_i, b_i \in R$ – коэффициенты уравнения прямой для -го направления.

Тогда множество параметров, определяющих положение отрезка -го направления можно определить следующим образом:

$$F_i = \{d_{i1}, d_{i2}, a_i, b_i\}, d_{i1} < d_{i2}, \tag{2.32}$$

где d_{i1}, d_{i2} – координаты абсцисс границ -го отрезка.

Пусть предикат P_i показывает, пересекается ли проекция отрезка A_jC_j, соединяющего диагонально противоположные точки ограничивающего прямоугольника некоторого объекта j – $A_j(x_{jA}; y_{jA})$ и $C_j(x_{jC}; y_{jC})$, и отрезок $d_{i1}d_{i2}$ для некоторого направления i:

$$P_{ij} = (d_{i1} \leq x_{jC} \wedge d_{i2} \leq x_{jA}). \tag{2.33}$$

Пусть предикат G_{ij} показывает, пересекает ли ограничивающий прямоугольник некоторого объекта j прямую, на которой лежит отрезок i:

$$G_{ij} = (y_{jA}y_{jC}(a_i x_{jA} + b_i)(a_i x_{jC} + b_i) > 0 \wedge$$
$$\wedge\, y_{jB}y_{jD}(a_i x_{jB} + b_i)(a_i x_{jD} + b_i) > 0). \tag{2.34}$$

Для решения поставленной задачи необходимо ввести предикат S_{ij}, который показывает, пересекает ли ограничивающий прямоугольник некоторого объекта j, отрезок некоторого направления i:

$$S_{ij} = (P_{ij} \wedge G_{ij}). \quad (2.35)$$

Пусть c_i^t – число транспортных средств, проехавших направление i за время t. S_{ij}^t – предикат, принимающий значение 1, если в момент времени t ограничивающий прямоугольник некоторого объекта j пересекает отрезок i. Тогда количество транспортных средств, проехавших -е направление за время $t + \tau$ определяется по следующему закону:

$$c_i^{t+\tau} = c_i^t + \sum_{j=1}^{m_{t+\tau}} S_{ij}^{t+\tau}, \quad (2.36)$$

где $m_{t+\tau}$ – количество обнаруженных объектов в момент времени $t + \tau$.

2.8 Выводы

В данном разделе была описана постановка задачи распознавания объектов на изображении, проанализированы и обработаны исходные данные для обучения нейронной сети YOLOv3, рассмотрен метод подсчета транспортных средств.

Описан метод обратного распространения ошибки, используемый для обучения нейронной сети. Рассмотрены функции потерь и функции активации. Были выбраны метрики для оценки качества работы нейронной сети. В частности, для оценки точности распознавания каждого класса обучающей выборки были выбраны метрики $precision$, $recall$ и IoU. Для оценки качества распознавания по всем классам была выбрана метрика mAP.

3 РЕАЛИЗАЦИЯ ПРОГРАММЫ И ПРОВЕРКА НА ТЕСТОВОМ ПЕРЕКРЕСТКЕ

3.1 Конфигурация и обучение нейронной сети YOLOv3

В таблице 3.1 представлена конфигурация нейронной сети. Она включает в себя 106 слоев и принимает на вход изображение с размерностью 960×480 пикселей.

Таблица 3.1 – Конфигурация нейронной сети YOLOv3

Кол-во	Тип слоя	Кол-во ядер	Размер/Шаг	Размерность на входе	Размерность на выходе
1	Convolutional	32	$3 \times 3/1$	$960 \times 480 \times 3$	$960 \times 480 \times 32$
	Convolutional	64	$3 \times 3/2$	$960 \times 480 \times 32$	$960 \times 480 \times 64$
	Convolutional	32	$1 \times 1/1$	$960 \times 480 \times 64$	$480 \times 240 \times 32$
	Convolutional	64	$3 \times 3/1$	$480 \times 240 \times 32$	$480 \times 240 \times 64$
1	Shortcut				
1	Convolutional	128	$3 \times 3/2$	$480 \times 240 \times 64$	$240 \times 120 \times 128$
2	Convolutional	64	$1 \times 1/1$	$240 \times 12 \times 128$	$240 \times 120 \times 64$
	Convolutional	128	$3 \times 3/1$	$240 \times 120 \times 64$	$240 \times 120 \times 128$
	Shortcut				
1	Convolutional	256	$3 \times 3/2$	$240 \times 12 \times 128$	$120 \times 60 \times 256$
8	Convolutional	128	$1 \times 1/1$	$120 \times 60 \times 256$	$120 \times 60 \times 128$
	Convolutional	256	$3 \times 3/1$	$120 \times 60 \times 128$	$120 \times 60 \times 256$
	Shortcut				
1	Convolutional	512	$3 \times 3/2$	$120 \times 60 \times 256$	$60 \times 30 \times 512$
8	Convolutional	256	$1 \times 1/1$	$60 \times 30 \times 512$	$60 \times 30 \times 256$
	Convolutional	512	$3 \times 3/1$	$60 \times 30 \times 256$	$60 \times 30 \times 512$
	Shortcut				
1	Convolutional	1024	$3 \times 3/2$	$60 \times 30 \times 512$	$30 \times 15 \times 1024$
4	Convolutional	512	$1 \times 1/1$	$30 \times 15 \times 1024$	$30 \times 15 \times 512$

Продолжение таблицы 3.1

Кол-во	Тип слоя	Кол-во ядер	Размер/Шаг	Размерность на входе	Размерность на выходе
	Convolutional	1024	$3 \times 3/1$	$30 \times 15 \times 512$	$30 \times 15 \times 1024$
	Shortcut				
3	Convolutional	512	$1 \times 1/1$	$30 \times 15 \times 1024$	$30 \times 15 \times 512$
	Convolutional	1024	$3 \times 3/1$	$30 \times 15 \times 512$	$30 \times 15 \times 1024$
1	Convolutional	68	$1 \times 1/1$	$30 \times 15 \times 1024$	$30 \times 15 \times 32$
	Yolo				
	Route				
	Convolutional	256	$1 \times 1/1$	$30 \times 15 \times 68$	$30 \times 15 \times 256$
	Upsample		$\times 2$	$30 \times 15 \times 256$	$60 \times 30 \times 256$
	Route				
3	Convolutional	256	$1 \times 1/1$	$60 \times 30 \times 256$	$60 \times 30 \times 256$
	Convolutional	512	$3 \times 3/1$	$60 \times 30 \times 256$	$60 \times 30 \times 512$
1	Convolutional	68	$1 \times 1/1$	$60 \times 30 \times 512$	$60 \times 30 \times 48$
	Yolo				
	Route				
	Convolutional	256	$1 \times 1/1$	$60 \times 30 \times 68$	$60 \times 30 \times 128$
	Upsample		$\times 2$	$60 \times 30 \times 128$	$120 \times 60 \times 128$
	Route				
3	Convolutional	128	$1 \times 1/1$	$120 \times 60 \times 128$	$120 \times 60 \times 128$
	Convolutional	256	$3 \times 3/1$	$120 \times 60 \times 128$	$120 \times 60 \times 256$
1	Convolutional	68	$1 \times 1/1$	$120 \times 60 \times 256$	$120 \times 60 \times 64$
	Yolo				

Столбец «Количество» показывает количество повторений различных комбинаций слоев. Столбец «Ядра» показывает количество синаптических ядер для сверточных слоев. Столбец «Размер/Шаг» показывает размер синап-

тического ядра и шаг, с которым применяется метод скользящего окна для сверточных слоев.

В процессе обучения нейронная сеть использует файлы с разметкой каждого изображения (рисунок 3.1).

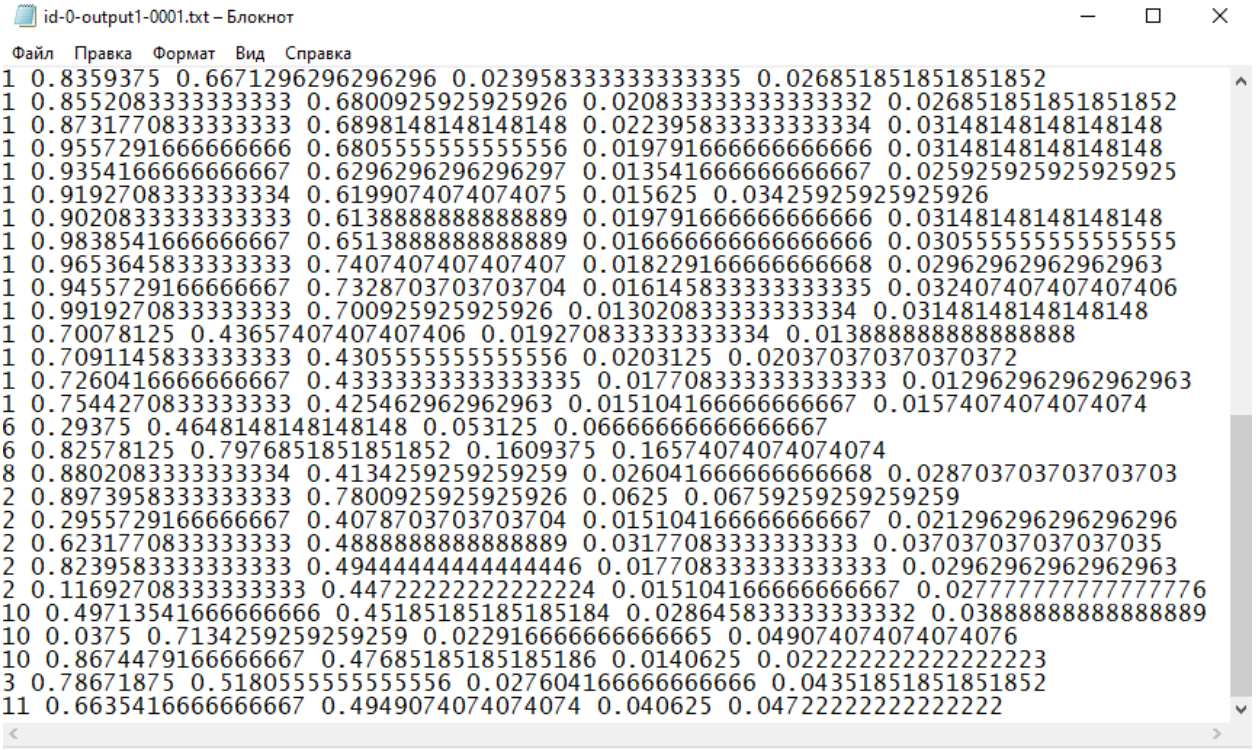

Рисунок 3.1 – Пример файла с разметкой для одного изображения

Каждая строка файла соответствует одному объекту на изображении и состоит из четырех чисел, разделенных пробелом. Первое число соответствует номеру класса соответствующего объекта. Второе и третье числа – координаты x, y левой верхней вершины ограничивающего прямоугольника, соответственно. Четвертое и пятое числа – длина и высота ограничивающего прямоугольника, соответственно. Координаты и размеры прямоугольников записаны в относительных, от изображения, размерах. Для их вычисления были использованы следующие формулы:

$$\tilde{x} = \frac{x - \frac{w_R}{2}}{w_I}, \tag{3.1}$$

$$\tilde{y} = \frac{y - \frac{h_R}{2}}{h_I}, \tag{3.2}$$

$$\tilde{w} = \frac{w_R}{w_I}, \tag{3.3}$$

$$\tilde{y} = \frac{h_R}{h_I}, \tag{3.4}$$

где x, y – координаты центра ограничивающего прямоугольника, измеряемые в пикселях;

$\tilde{x}\tilde{y}$ – относительные координаты верхней левой вершины ограничивающего прямоугольника;

w, h – размеры ограничивающего прямоугольника, измеряемые в пикселях;

$\tilde{w}\tilde{h}$ – относительные размеры ограничивающего прямоугольника.

Для обучения нейронной сети было установлено 10000 итераций. Блок-схема алгоритма обучения представлена на рисунке 3.2.

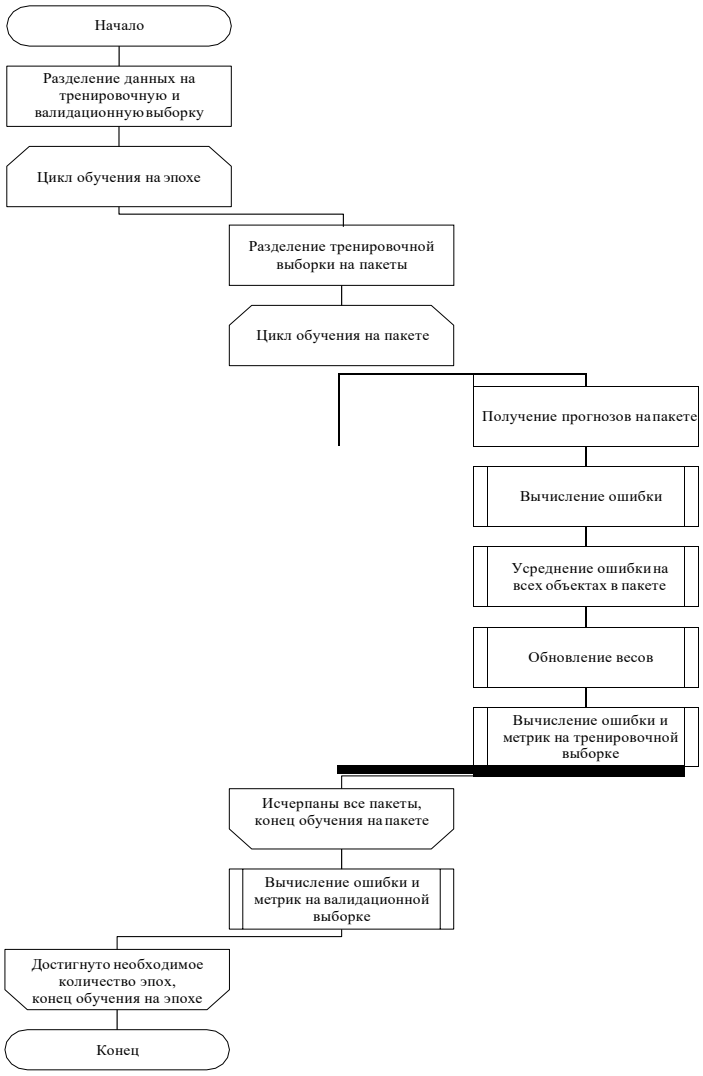

Рисунок 3.2 – Алгоритм обучения нейронной сети

Перед запуском процесса обучения происходит разделение входных данных на обучающую и тестовую выборки в соотношении 80/20%, соответственно. Для обучения используется метод пакетного градиентного спуска.

Для каждой эпохи обучения происходит разделение обучающей выборки на пакеты, в каждом из которых по 4 изображения. Для каждого изображения из пакета происходит прямой ход и вычисляется значение функции потерь. Обратное распространение ошибки происходит только после обработки всех изображений и усреднения функции потерь на всем пакете.

Каждые 300 эпох происходит сохранение полученных весов и пересчет метрик качества на тестовой выборке. Периодичное сохранение весов позволяет после обучения получить только те веса, которые показали наилучшие показатели точности на тестовой выборке.

3.2 Результаты обучения и полученные метрики качества

В результате обучения были получены графики изменения функции потерь (рисунок 3.3) и метрики mAP (рисунок 3.4). На приведенных рисунках можно увидеть, как значение метрики mAP в процессе обучения растет, а значение функции потерь – падает. Итоговое значение функции потерь составило 0.1529. Значение метрики mAP = 0.89.

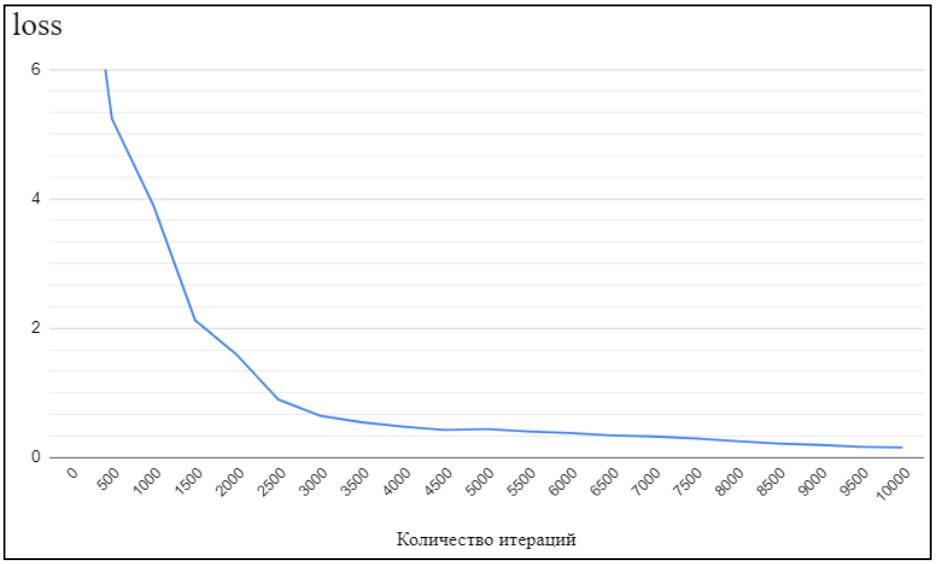

Рисунок 3.3 – График изменения значения функции потерь в процессе обучения

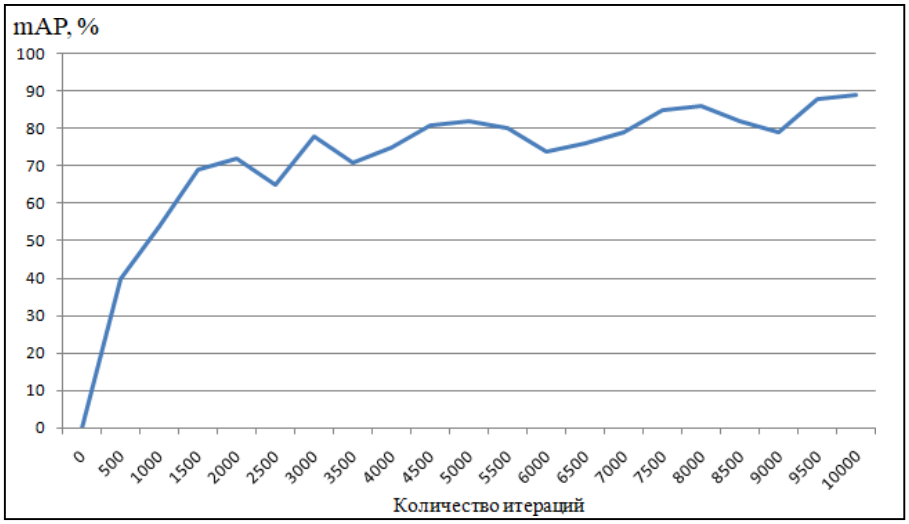

Рисунок 3.4 – График изменения метрики mAP
в процессе обучения

На основе данных, полученных при обработке видео, были вычислены значения метрик precision и recall для каждого класса (таблица 3.2).

Таблица 3.2 – Значения метрик precision и recall для каждого класса

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	0.7
Троллейбус	0.71	0.7
Спец. Транспорт	0.72	0.68
Трамвай	0.92	0.95
Автопоезд	0.9	0.96

Проанализировав полученные данные можно заметить, что для классов с наименьшим количеством объектов в обучающей выборке были получены наихудшие показатели точности и полноты обнаружения. В качестве исключения из этого правила можно выделить классы «Трамвай» и «Автопоезд».

На рисунке 3.5 изображен пример работы нейронной сети.

Рисунок 3.5 – Пример работы нейронной сети

3.3 Алгоритм подсчета транспортных средств

В целях предотвращения ложных обнаружений, для тестового перекрестка была реализовано изображение маски, позволяющее оставить только необходимую область на изображении (рисунок 3.6).

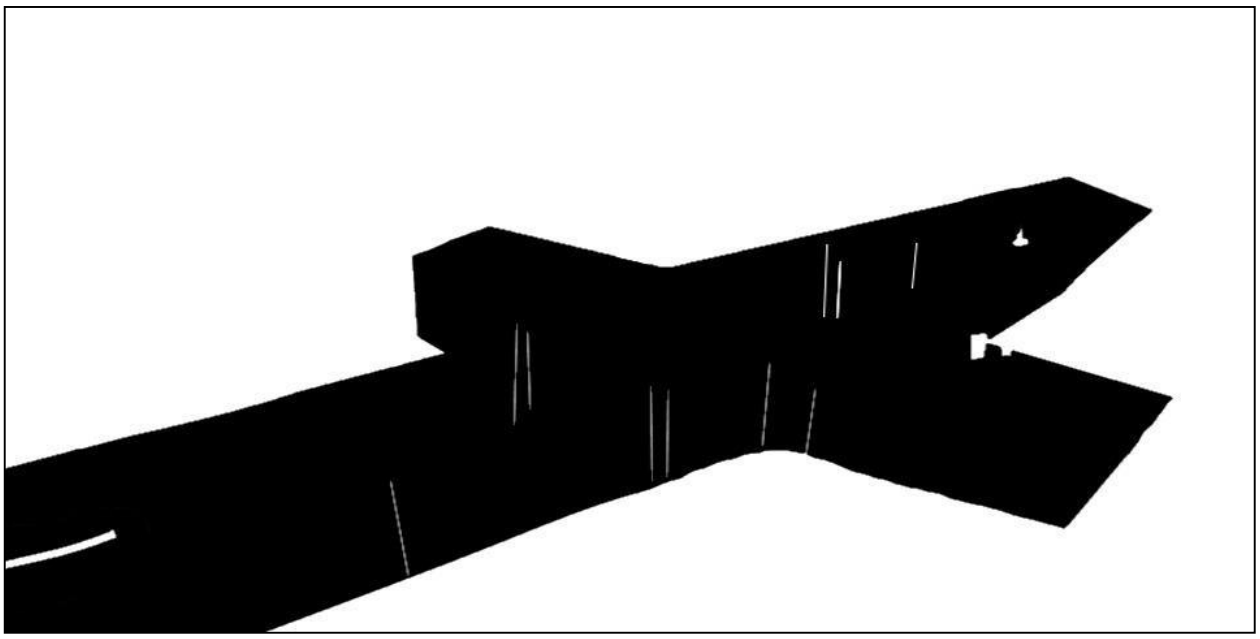

Рисунок 3.6 – Изображение маски

Маска представляет собой матрицу M, элементы которой это числа 0 либо 255, 0 соответствует черному пикселю, а 255 – белому. Изображение представимо в виде набора трех матриц, каждая из которых соответствует красному, зеленому либо синему цветовому каналу, а их элементы – числа из диапазона $[0; 255]$. Тогда необходимое изображение (рисунок 3.7) можно получить следующим образом:

$$b_{ij}^{R} = \min(a_{ij}^{R} + m_{ij}, 255), \quad (3.5)$$

$$b_{ij}^{G} = \min(a_{ij}^{G} + m_{ij}, 255), \quad (3.6)$$

$$b_{ij}^{B} = \min(a_{ij}^{B} + m_{ij}, 255), \quad (3.7)$$

где $b_{ij}^{R}, b_{ij}^{G}, b_{ij}^{B}$ – ij-ые элементы матриц красного, зеленого, синего цветовых каналов итогового изображения, соответственно;

$a_{ij}^{R}, a_{ij}^{G}, a_{ij}^{B}$ – ij-ые элементы матриц красного, зеленого, синего цветовых каналов исходного изображения, соответственно;

m_{ij} – ij-ый элемент матрицы маски;

w, h – длина и высота исходного и итогового изображения;

$i = \overline{1,h}, j = \overline{1,w}$

Рисунок 3.7 – Изображение с наложенной маской

Для подсчета транспортных средств на тестовом перекрестке было реализовано 8 отрезков, изображенных на рисунке 3.9. Зеленым цветом изображены отрезки подсчета для направлений въезда на перекресток, оранжевым – направления выезда с перекрестка.

При пересечении отрезка ограничивающим прямоугольником, соответствующий ему счетчик инкрементируется. Блок-схема алгоритма подсчета для n направлений изображена на рисунке 3.8.

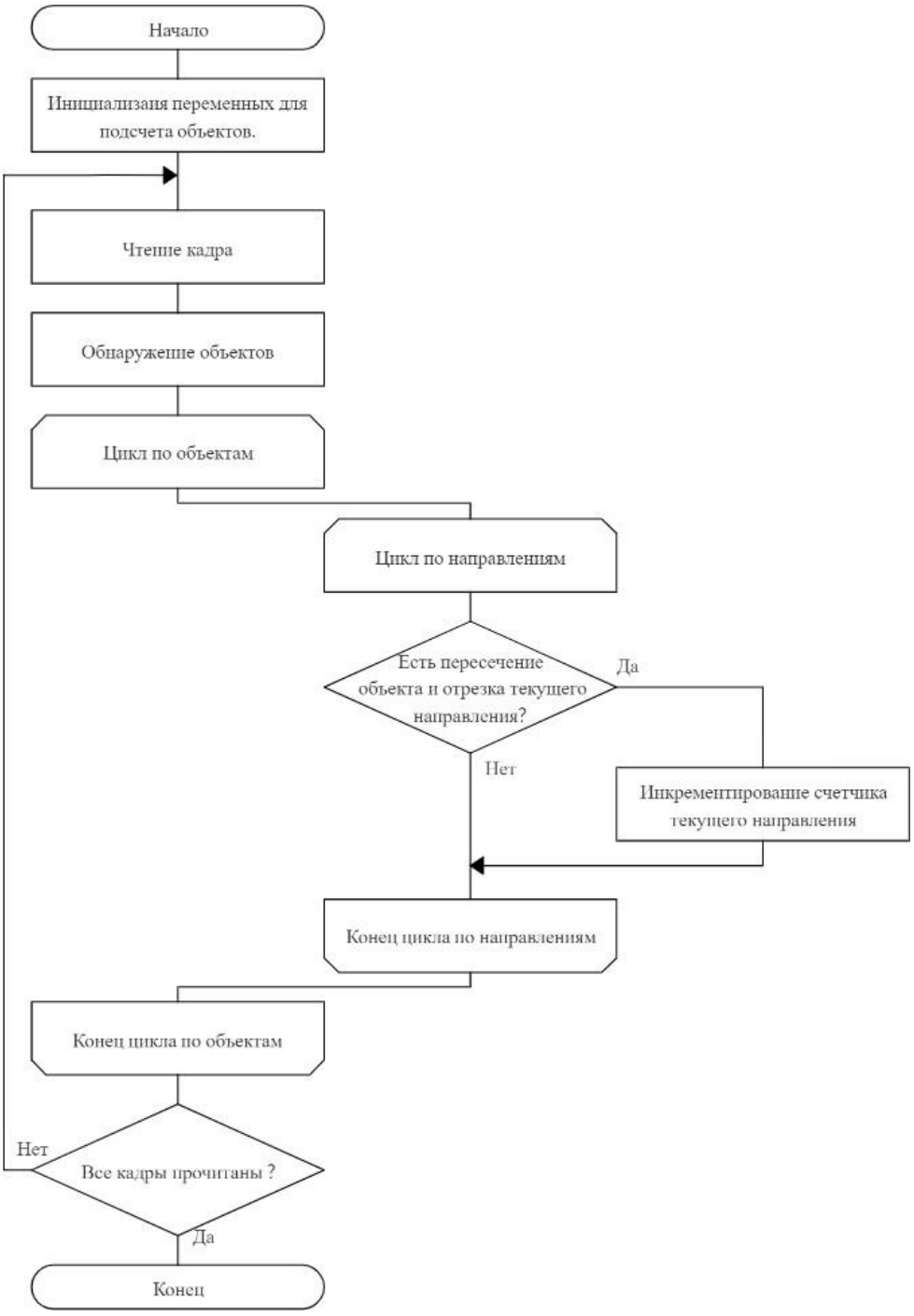

Рисунок 3.8 – Алгоритм подсчета транспортных средств

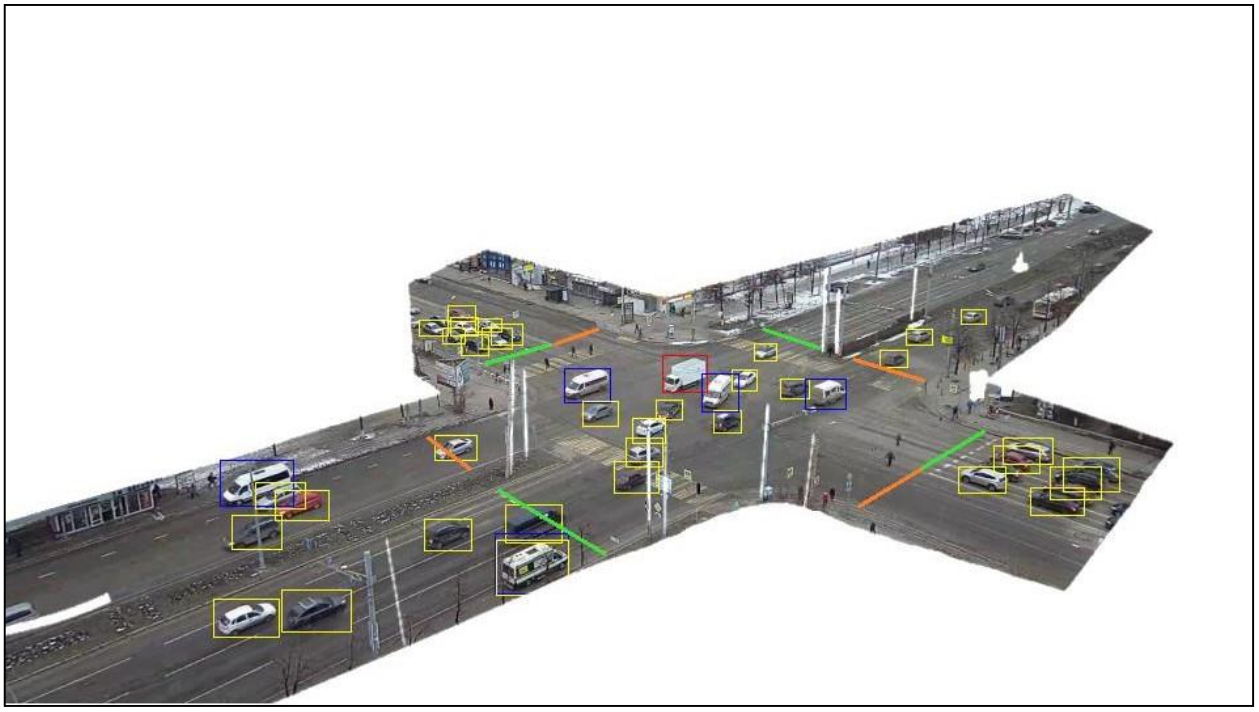

Рисунок 3.9 – Отрезки, обозначающие въезд и выезд с перекрестка

3.4 Подсчет транспортных средств в режиме реального времени

Для тестирования программы в режиме реального времени был выбран тестовый перекресток (рисунок 3.10) с 4 направлениями въезда и 4 направлениями выезда.

Рисунок 3.10 – Тестовый перекресток

Для оценки точности подсчета транспортных средств было сформировано четыре 15-ти минутных видеофрагментов в разное время суток и выполнен ручной подсчет транспорта в них. Результаты сравнения данных представлены на рисунке 3.11 в виде графиков разного цвета. Красным цветом представлен график, построенный на данных, полученных в результате работы программы. Синим – данные, основанные на ручном подсчете транспортных средств.

Максимальная ошибка подсчета равна 5 транспортным средствам в минуту при общем количестве транспортных средств равным, в среднем, 64. Таким образом, ошибка не превысила 8% от общего количества транспортных средств.

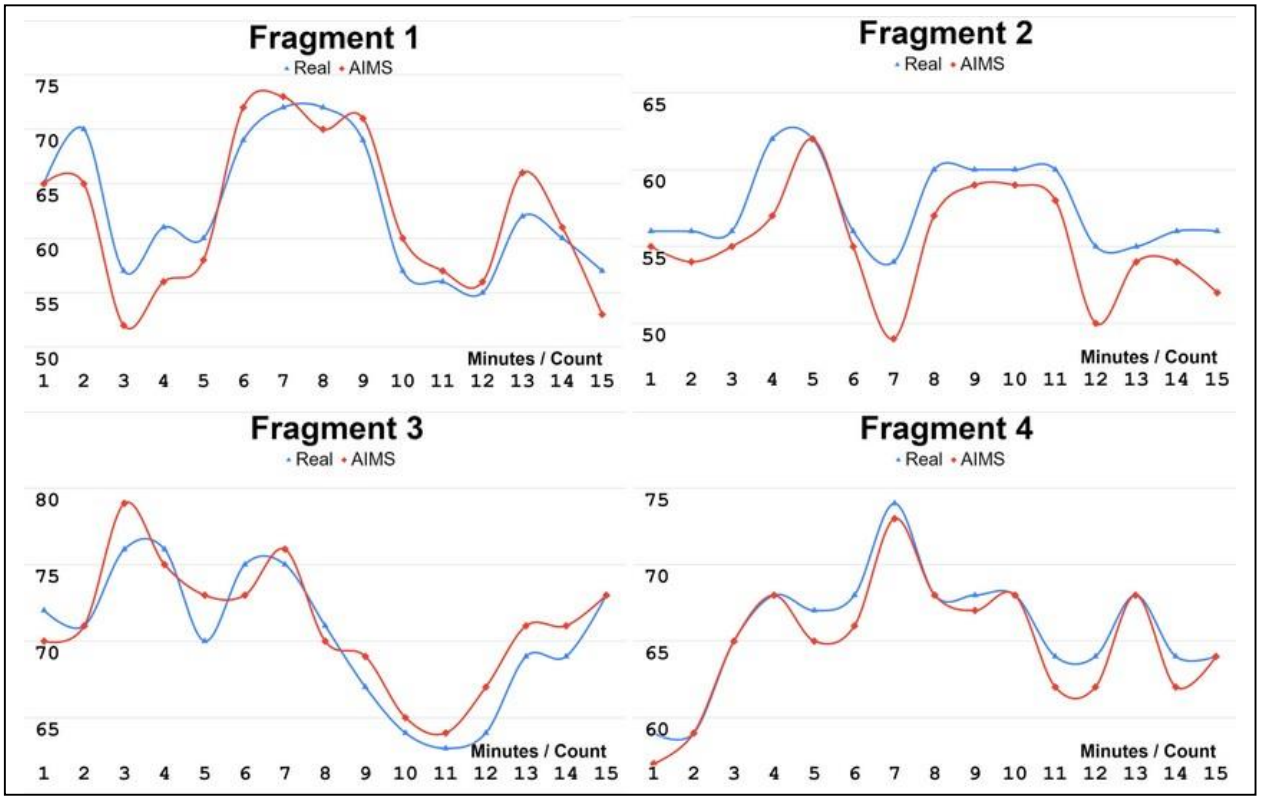

Рисунок 3.11 – Результатов подсчета транспортных средств. Красным цветом обозначен подсчет программы, синим – ручной подсчет

3.5 Проект «AIMS»

AIMS – проект, целью которого является разработка и внедрение системы мониторинга дорожных потоков с использованием камер уличного

наблюдения. Данный проект специализируется на сборе следующих параметров дорожного потока:

– средняя скорость транспортных средств;

– количество выбросов загрязняющих веществ в атмосферу потоками транспортных средств;

– интенсивность потока транспортных средств;

– пропускная способность дорожных узлов.

Демонстрация работы данной системы представлена на рисунке 3.12.

Рисунок 3.12 – Демонстрация работы системы AIMS

Архитектура системы AIMS (рисунок 3.13) состоит из 6 основных процессов:

– чтение кадров из видеопотока;

– обнаружение транспортных средств на изображении;

– отслеживание транспортных средств;

– вычисление географических координат для каждого транспортного средства;

– вычисление скоростей транспортных средств;

– вычисление количества выбросов вредных веществ.

Результаты данной работы были успешно внедрены и используются в рамках этого проекта.

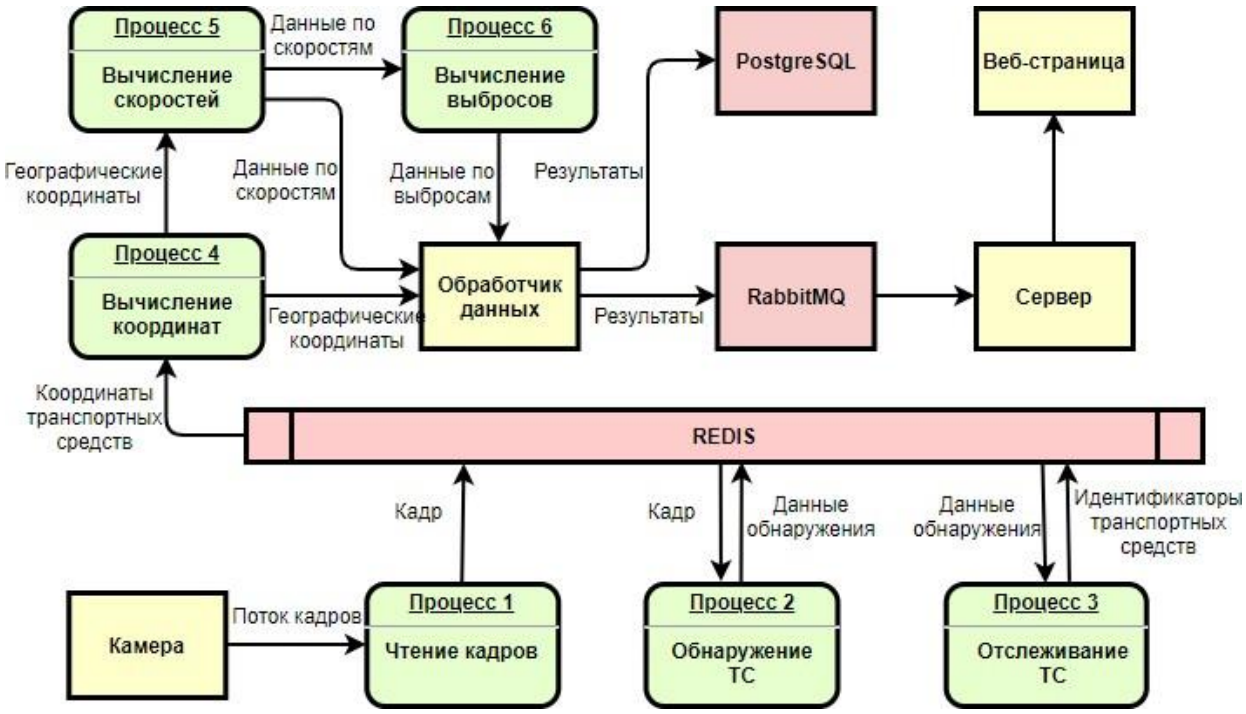

Рисунок 3.13 – Архитектура системы AIMS

3.6 Выводы

В данном разделе была обучена нейронная сеть и были реализованы программные модули, описанные в главе 2. В результате обучения были получены метрики качества, показывающие точность обнаружения каждого из классов транспортных средств. Был сделан вывод о зависимости количества объектов каждого класса в обучающей выборке и итоговой точностью обнаружения.

Программа по подсчету транспортных средств была проверена на тестовых данных. Максимальная ошибка подсчета не превышает 8% от общего количества транспортных средств на тестовом перекрестке.

ЗАКЛЮЧЕНИЕ

Цель данной работы состояла в разработке системы, позволяющей получать данные о структуре транспортного потока, направлениях движения и количестве транспортных средств в режиме реального времени.

В соответствии с целью, в первой главе данной работы был проведен обзор существующих решений в области мониторинга транспортных потоков. Рассмотрены библиотеки, предоставляющие упрощенный интерфейс для работы с нейронными сетями. Проанализированы нейросетевые подходы и существующие модели в задачах обнаружения объектов на изображении. Были выявлены достоинства и недостатки нейросетевых моделей в рамках поставленной задачи и на основании этого был сделан выбор подходящей архитектуры нейронной сети.

Во второй главе была сформулирована постановка задачи обнаружения объектов на изображении, описана архитектура нейронной сети. Рассмотрен метод обучения нейронной сети, ее функции активации и функция потерь. Выбраны и описаны метрики для оценки качества работы нейронной сети. Представлен метод подсчета транспортных средств.

В третьей главе представлены алгоритмы обучения нейронной сети и подсчета транспортных средств. Была обучена нейронная сеть и получены значения метрик качества. Также программа была успешно протестирована на тестовом перекрестке в режиме реального времени.

Результаты данной работы успешно используются в рамках проекта AIMS для решения задачи мониторинга транспортных потоков. В дальнейшем планируется расширить функционал программы для одновременного мониторинга нескольких дорожных узлов.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Зайченко, В. Машинное обучение против фрода / В. Зайченко, М. Земскова // Открытые СУБД [Электронный ресурс]. – Дата обновления: 26.05.2017. URL: https://www.osp.ru/os/2017/02/13052223/ (дата обращения: 16.02.2020).

2 Шолле, Ф. Глубокое обучение на Python / Ф. Шолле. – Санкт-Петербург: Питер, 2018. – 400 с.

3 Сурцков, М. Автоэнкодеры в Keras /М. Сурцков [Электронный ресурс]. – Дата обновления: 23.06.2017 URL: https://habr.com/ru/post/331382/ (дата обращения: 16.02.2020).

4 Радченко, В. Открытый курс машинного обучения. Тема 5. Композиции: бэггинг, случайный лес / В. Радченко [Электронный ресурс]. – Дата обновления: 27.03.2017 URL: https://habr.com/ru/company/ods/blog/324402/ (дата обращения: 15.04.2020).

5 Царьков, С.В. Различные стратегии сэмплинга в условиях несбалансированности классов / С.В. Царьков [Электронный ресурс]. URL: https://basegroup.ru /community/articles/imbalance-datasets (дата обращения: 21.02.2019).

6 Кибзун, А.И. Теория вероятностей и математическая статистика. Базовый курс с примерами и задачами / А.И. Кибзун, Е.Р. Горяинова. Издательство ФИЗМАЗЛИТ, 2002. – 34 с.

7 Гасников, А.В. Современные численные методы оптимизации. Метод универсального градиентного спуска / А.В. Гасников. МФТИ, Москва, 2018. – 21с.

8 Уоссермен, Ф. Нейрокомпьютерная техника: Теория и практика / Ф. Уоссермен. Издательство Мир, 2002. – 16 с.

9 Gibiansky, A. Recurrent Neural Networks / A. Gibiansky [Электронный ресурс]. – Дата обновления: 21.03.2014. URL: http://andrew.gibiansky.com/blog/machine-learning/recurrent-neural-networks/ (дата обращения: 17.02.2020).

10 Ren, S. Faster R-CNN: towards real-time object detection with region proposal networks. IEEE Trans Pattern Anal Mach Intell / Ren, S [Электронный ресурс]. – Дата обновления: 06.01.2016. URL: https://arxiv.org/abs/1506.01497 (дата обращения: 08.04.2020).

11 Breiman, L. Classification and regression trees / L. Breiman, J. Friedman, R. Olshen, C. Stone. – New-York: Taylor & Francis, 1983. – 368 p.

12 Komiya, K. Negation Naive Bayes for Categorization of Product Pages on the Web/ K. Komiya, N. Sato, K. Fujimoto, Y. Kotani // Proceedings of the International Conference Recent Advances in Natural Language Processing, 2011. – P. 586–591.

13 Cilimkovic, M. Neural Networks and Back Propagation Algorithm / M. Cilimkovic. Institute of Technology Blanchardstown Blanchardstown Road North, Dublin 15, 2010. – 7 p.

14 Rojas, R. Neural Networks A Systematic Introduction / R. Rojas. SpringerVerlag, Berlin, 1996 – 184 p.

15 Kovalczyk, A. Support vector machines succinctly / A. Kovalczyk. Syncfusion, 2017. – 12 p.

16 Vapnik, V.N. Support-Vector Networks / V.N. Vapnik, C. Cortes. Machine Learning, 20, 273–297, 1995. – 273 p.

17 Vapnik, V.N. On a perceptron class / V.N. Vapnik, A.Ya. Chervonenkis. Avtomat. i Telemekh., Volume 25, Issue 1, 1964. – 112 p.

18 Myers, A. Im2Calories: towards an automated mobile vision food diary / A. Myers, N. Johnston Google – 8 p.

19 Cloud AutoML [Электронный ресурс]. URL: https://cloud.google.com/automl/ (дата обращения: 16.03.2020).

20 Nvidia. Nvidia Tesla K80 S [Электронный ресурс]. URL: https://www.nvidia.com/ru-ru/data-center/tesla-k80/ (дата обращения: 20.03.2020).

21 University of Toronto S [Электронный ресурс]. URL: https://www.cs.toronto.edu/~kriz/cifar.html (дата обращения: 25.03.2020).

22 Wei, M. An Equivalence of Fully Connected Layer and Convolutional Layer / M. Wei, L. Jun. Department of Computer Science, 2017. – 1 p.

23 Masci, J. A fast learning algorithm for image segmentation with max-pooling / J. Masci, A. Giusti. Convolutional networks, 2013. – 2 p.

24 Bing, Xu, Empirical Evaluation of Rectified Activations in Convolution Network / Xu Bing, W. Naiyan. Arxiv, 2015. – 2 p.

25 Clevert, D. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs) / D. Clevert, T. Unterthiner. Machine Learning, 2016. – 5 p.

26 Martins, A.F.T. From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification / A.F.T. Martins, R.F. Astudillo. Arxiv, 2016. – 2 p.

27 Reddit. AMA: We are the Google Brain team. We'd love to answer your questions about machine learning, 2017 S [Электронный ресурс]. URL:https://www.reddit.com/r/MachineLearning/comments/4w6tsv/ama_we_are_the_google_brain_team_wed_love_to/ (дата обращения: 27.03.2020).

28 Ioffe, S. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift / S. Ioffe, C. Szegedy. Arxiv, 2015. – 3 p.

29 Merity, S. Regularizing and Optimizing LSTM Language Models / S. Merity, R. Socher. Arxiv, 2016. – 4 p.

30 Vedaldi, A. Convolutional Neural Networks for MATLAB / A. Vedaldi, K. Lenc. MatConvNet, 2016. – 22 p.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ 1

Текст программы для аугментации изображений

```
import numpy as np
import imgaug as ia
import imgaug.augmenters as iaa
from imgaug.augmentables.bbs import BoundingBox, BoundingBoxesOnImage
import cv2
import  os
import threading



outputDir = 'output_aug/'
prefix = 'output_'

_, src, num = sys.argv
inputDir = src
outputDir = outputDir + prefix + num + '/'
threadCount = 36
readExt = ".jpg"

def convert(ind, name, seq, image, boxes, types, width, height):
    #конвертирование
    bbs = BoundingBoxesOnImage(boxes, shape=image.shape)

    image_aug, bbs_aug = seq(image=image, bounding_boxes=bbs)

    #запись рамок в файл
    textFile = open(outputDir + name + "-aug" + str(ind) + ".txt",
"w")
    for i, box in enumerate(bbs_aug):
```

```
        if box.x1 <= 0 or box.y1 <= 0 or box.x2 >= width or box.y2 >=
height:
            print(box.x1, box.y1, box.x2, box.y2)
        w = box.x2 - box.x1
        h = box.y2 - box.y1
        cx = box.x1 + w / 2
        cy = box.y1 + h / 2
        textFile.write(types[i] + " " + str(cx / width) + " " +  str(cy
/ height) + " " + str(w / width) + " " + str(h / height)+"\n")
    textFile.close()

    #сохранение картинки
    cv2.imwrite(outputDir + name + "-aug" + str(ind) + ".jpg", im-
age_aug)


def convertBackground(threadInd, names, seq):
    for imageInd, name in enumerate(names):
        try:
            #считывается картинка
            image = cv2.imread(inputDir + name + readExt)
            height, width, channels = image.shape

            #считываются рамки
            textFile = open(inputDir + name + ".txt", "r")
            lines = textFile.readlines()
            textFile.close()

            #в types сохраняются типы объектов, а boxes объекты
BoundingBox с координатами рамки
            types = []
            boxes = []
            for line in lines:
                words = line.replace('\n', '').split()
```

```
                cx = float(words[1]) * width
                cy = float(words[2]) * height
                w = float(words[3]) * width
                h = float(words[4]) * height
                x1 = int(cx - w / 2)
                y1 = int(cy - h / 2)
                x2 = int(cx + w / 2)
                y2 = int(cy + h / 2)
                boxes.append(BoundingBox(x1=x1, y1=y1, x2=x2, y2=y2))
                types.append(words[0])

            #конвертирование 12 раз для каждой картинки
            count = 0
            for i in range(12):
                try:
                    convert(i, name, seq, image, boxes, types, width,
height)
                    count += 1
                except:
                    print("convert error", threadInd, ":", i, name)
                    pass

            print("saved", threadInd, ":", imageInd, "(", count, ")",
name)

        except:
            print("error", threadInd, ":", name)
            pass
    print("end", threadInd)


def main():
    global threadCount
```

```
    #настройки
    seq = iaa.SomeOf((3, 4), [ #от 3 до 4
        iaa.Fliplr(0.5), #отзеркалить по горизонтали с вероятностью
0.5
        iaa.OneOf([ #перспективное и аффинное преобразования
            iaa.PerspectiveTransform(scale=(0.02, 0.06)),
            iaa.SomeOf((1, 3), [
                iaa.Affine(scale={"x": (0.95, 1.05), "y": (0.95,
1.05)}),
                iaa.Affine(translate_percent={"x": (-0.02, 0.02), "y":
(-0.02, 0.02)}),
                iaa.Affine(rotate=(-1, 1))
            ])
        ]),
        iaa.OneOf([ #размытия, черные точки, дождь, снег, шум
            iaa.GaussianBlur((1, 1.5)),
            iaa.AverageBlur((1, 4)),
            iaa.MedianBlur((1, 5)),
            iaa.Sharpen(alpha=(0.1, 0.2), lightness=(0.1, 0.2)),
            iaa.Emboss(alpha=(0.2, 0.4), strength=(0.4, 0.6)),
            iaa.Dropout((0.02, 0.05), per_channel=0.5),
            iaa.AveragePooling([1, 2]),
            iaa.CoarseDropout((0.009, 0.01), size_percent=(0.9, 0.9),
per_channel=0.2),
            iaa.ElasticTransformation(alpha=(0.5, 0.9), sigma=0.25),
            iaa.Rain(speed=(0.3, 0.6)),
            iaa.Snowflakes(flake_size=(0.4, 0.7), speed=(0.02, 0.05)),
            iaa.AdditiveGaussianNoise(loc=0.5, scale=(10, 15),
per_channel=0.5)
        ]),
        iaa.OneOf([ #преобразования цветов
            iaa.Add((-20, 20), per_channel=0.5),
            iaa.AddToHueAndSaturation((-20, 20)),
            iaa.Multiply((1.2, 1.5), per_channel=0.5),
```

```
            iaa.LinearContrast((1, 2), per_channel=0.5),
            iaa.Grayscale(alpha=(0.5, 1.0))
        ])
    ], random_order=True)

    #достаются все файлы из папки и выбираются только jpg
    fileNames = os.listdir(inputDir)
    imageNames = []
    for fileName in fileNames:
        name, extension = os.path.splitext(fileName)
        if extension == readExt:
            imageNames.append(name)

    #имена разбиваются по потокам
    namesByThreads = []
    for i in range(threadCount):
        count = int(len(imageNames) / threadCount)
        if count > 0:
            namesByThreads.append(imageNames[0:count])
            del imageNames[0:count]
        threadCount -= 1
    for i, namesByThread in enumerate(namesByThreads):
        print("thread", i, "-", len(namesByThread))
    #создаются потоки
    for i, namesByThread in enumerate(namesByThreads):
        thread       =       threading.Thread(target=convertBackground,
kwargs={"threadInd": i, "names": namesByThread, "seq": seq})
        thread.daemon = True
        thread.start()

    input()
if __name___== "_main_":
    main()
```

ПРИЛОЖЕНИЕ 2

Текст программы для получения кадров с видеопотока

```
import cv2
import numpy
import pika
import redis
import time
import pickle
import json


class DataGetter:

    def_init_(self, module_params, crossroad_params):
        self.id = str(crossroad_params['id'])
        self.url = str(crossroad_params['video_url'])
        self.frame_rate = int(crossroad_params['frame_rate'])

        self.calibration_frame_width                                    =
int(module_params['calibration_frame_width'])
        self.calibration_frame_height                                   =
int(module_params['calibration_frame_height'])
        self.neural_network_frame_width                                 =
int(module_params['neural_network_frame_width'])
        self.neural_network_frame_height                                =
int(module_params['neural_network_frame_height'])

        calibration_data                                                =
numpy.load(str(crossroad_params['calibration_matrix']),             al-
low_pickle=True)
        mtx = calibration_data[0]
        dist = calibration_data[1]
```

```
        size = (self.calibration_frame_width,
self.calibration_frame_height)
        new_matrix, roi = cv2.getOptimalNewCameraMatrix(mtx, dist,
size, 0, size)
        self.map1, self.map2 = cv2.initUndistortRectifyMap(mtx, dist,
None, new_matrix, size, cv2.CV_32FC1)

        self.queue_to =
"{}::{}".format(str(module_params['queue_to']), self.id)
        self.connection = pi-
ka.BlockingConnection(pika.ConnectionParameters('localhost'))
        self.channel = self.connection.channel()
        self.channel.queue_declare(queue=self.queue_to, argu-
ments=module_params['queue_to_params'])

        self.mask = cv2.imread(str(crossroad_params["mask"]))
        self.mask = cv2.resize(self.mask,
(self.calibration_frame_width, self.calibration_frame_height))

        self.redis = redis.Redis()

        self.capture = None
        self.frame_skip_count = None
        self.init_video_capture()

    def start(self):
        frame_ind = 0
        while True:
            try:
                if not self.capture.isOpened():
                    self.init_video_capture()
                    continue
                success, frame_read = self.capture.read()
                if frame_read is None or not success:
```

```
                self.init_video_capture()
                continue
            except:
                self.init_video_capture()
                continue

            frame_ind += 1
            if frame_ind < self.frame_skip_count:
                continue
            frame_ind = 0

            millis = int(round(time.time() * 1000))

            frame_resized = cv2.resize(frame_read,
(self.calibration_frame_width, self.calibration_frame_height))

            frame_undistorted = self.undistort(frame_resized)

            frame_masked = cv2.addWeighted(frame_undistorted, 1,
self.mask, 1, 0, 3)

            frame_for_neural = cv2.resize(frame_masked,
(self.neural_network_frame_width, self.neural_network_frame_height))

            self.redis.setex("{}::{}_big".format(self.id, millis), 10,
value=pickle.dumps(frame_undistorted))
            self.redis.setex("{}::{}_neural".format(self.id, millis),
10, value=pickle.dumps(frame_for_neural))

            self.channel.basic_publish(exchange="", rout-
ing_key=self.queue_to, body=json.dumps({
                "id": self.id,
                "millis": millis
            }))
```

```
    def init_video_capture(self):
        self.capture = cv2.VideoCapture(self.url)
        fps = self.capture.get(cv2.CAP_PROP_FPS)
        self.frame_skip_count = fps / self.frame_rate

    def undistort(self, image):
        return cv2.remap(image, self.map1, self.map2, interpola-
tion=cv2.INTER_LINEAR)

    def close(self):
        self.connection.close()
```

ПРИЛОЖЕНИЕ 3

Текст программы для обнаружения транспортных средств

```
import pika
import modules.darknet.darknet as darknet
import json
import cv2
import os
import redis
import time
import pickle


class Networker:

    def_init_(self, module_params, crossroad_params):
        self.id = str(crossroad_params['id'])

        self.darknet_cfg_file = str(module_params['darknet_cfg_file'])
        self.darknet_data_file                                          =
str(module_params['darknet_data_file'])
        self.darknet_weights_file                                       =
str(module_params['darknet_weights_file'])

        self.queue_from                                                 =
"{}::{}".format(str(module_params['queue_from']), self.id)
        self.queue_to                                                   =
"{}::{}".format(str(module_params['queue_to']), self.id)
        self.connection                      =                      pi-
ka.BlockingConnection(pika.ConnectionParameters('localhost'))
        self.channel = self.connection.channel()
        self.channel.queue_declare(queue=self.queue_from,         argu-
ments=module_params['queue_from_params'])
```

```
        self.channel.queue_declare(queue=self.queue_to, argu-
ments=module_params['queue_to_params'])

        self.redis = redis.Redis()

        if not os.path.exists(self.darknet_cfg_file):
            raise ValueError("Invalid config path " +
os.path.abspath(self.darknet_cfg_file))
        if not os.path.exists(self.darknet_data_file):
            raise ValueError("Invalid data file path " +
os.path.abspath(self.darknet_data_file))
        if not os.path.exists(self.darknet_weights_file):
            raise ValueError("Invalid weight path " +
os.path.abspath(self.darknet_weights_file))

        self.net_main = dark-
net.load_net_custom(self.darknet_cfg_file.encode("ascii"),
self.darknet_weights_file.encode("ascii"), 0, 1)
        self.meta_main = dark-
net.load_meta(self.darknet_data_file.encode("ascii"))

        self.network_width = darknet.network_width(self.net_main)
        self.network_height = darknet.network_height(self.net_main)
        self.darknet_image = darknet.make_image(self.network_width,
self.network_height, 3)

    def start(self):
        while True:
            try:
                msg = self.channel.basic_get(queue=self.queue_from,
auto_ack=True)
                if msg is None:
                    time.sleep(0.01)
                else:
```

```
                    body = msg[2]
                    if body is not None:
                        data = json.loads(body.decode())
                        image_bytes                                          =
self.redis.get("{}::{}_neural".format(data["id"], data["millis"]))
                        image = pickle.loads(image_bytes)
                        self.detector(image, data)
            except:
                pass

    def detector(self, image, data):
        frame_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        frame_resized       =       cv2.resize(frame_rgb,       (dark-
net.network_width(self.net_main),                                    dark-
net.network_height(self.net_main)),
                                    interpolation=cv2.INTER_LINEAR)
        darknet.copy_image_from_bytes(self.darknet_image,
frame_resized.tobytes())
        detections        =        darknet.detect_image(self.net_main,
self.meta_main, self.darknet_image, thresh=0.25)

        converted_detections = []
        for detection in detections:
            x, y, w, h, score, obj_type = float(detection[2][0]),
float(detection[2][1]), float(detection[2][2]),\
                                        float(detection[2][3]),   de-
tection[1], detection[0]
            converted_detections.append(
                {
                    "x": x / self.network_width,
                    "y": y / self.network_height,
                    "w": w / self.network_width,
                    "h": h / self.network_height,
                    "score": detection[1],
```

```
                    "class": detection[0].decode()
                })

        data["detections"] = converted_detections
        self.channel.basic_publish(exchange="", routing_key=self.queue_to, body=json.dumps(data))

    def close(self):
        self.connection.close()
```