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.

Обнаружение и подсчет пешеходов по всем направлениям дорожного узла в видеопотоке реального времени с помощью сверточной нейронной сети

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ–01.03.02.2020.052.ПЗ ВКР

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

Челябинск
2020

АННОТАЦИЯ

Алиева А.А. Обнаружение и подсчет пешеходов по всем направлениям дорожного узла в видеопотоке реального времени с помощью сверточной нейронной сети. – Челябинск: ЮУрГУ, ЕТ-412, 68 с., 42 ил., 4 табл., библиогр. список – 32 наим.

Целью данной работы является разработка модели искусственной нейронной сети, которая в режиме реального времени обнаруживает и ведет подсчет пешеходов в каждом направлении дорожного узла. Решается задача детекции объектов на видеопоследовательности, предлагается нейросетевой метод – использование модели нейронной сети YOLOv3.

Модель данных нейронной сети реализована на языке Си с использованием фреймворка DarkNet. Для подготовки и обработки данных используется платформа Sypervisely. Проведен анализ работы модели с помощью различных метрик качества распознавания. Реализован подсчет пешеходов в режиме реального времени и проведен анализ точности подсчета.

Реализованная модель может применяться в городских транспортных службах для распознавания объектов на видеопоследовательностях в режиме реального времени.

ОГЛАВЛЕНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

CNN – сверточная нейронная сеть.

GPU (англ. Graphics Processing Units) – графический процессор.

mAP (англ. mean Average Precision) – одна из основных метрик оценки качества ранжирования.

ВВЕДЕНИЕ

Согласно данным Всемирной организации здравоохранения (ВОЗ) на 2015 год, вследствие дорожно-транспортных происшествий (ДТП) ежегодно погибают более 1,25 миллиона человек (186 тысяч из них дети). От 20 млн. до 50млн. получают различные травмы [1]. Ежедневно погибают более 3 тысяч человек и около 100 тысяч получают серьезные травмы. Этот показатель практически не меняется с 2007 года.

Основной причиной смерти людей от 15 до 29 лет являются дорожные аварии. Наибольший процент смертности наблюдается среди мотоциклистов (23% всех случаев смерти в результате ДТП), пешеходов (22%) и велосипедистов (4%). В 90% случаев ДТП со смертельным исходом происходит в странах с низким и средним уровнем дохода.

В России по статистике Министерства внутренних дел (МВД) в 2018 году произошло 168 тысяч ДТП с пострадавшими [2]. Из них 50 тысяч с участием пешеходов. Каждое девятое ДТП закончилось смертельным исходом. За год на дорогах страны погибло около 18 тысяч человек. Еще 215 тысяч человек получили различные травмы.

Сравнивая статистику ДТП с участием пешеходов, согласно данным МВД, за первое полугодие 2019 года произошло 20,4 тыс. ДТП (21,2 тысячи за аналогичный период 2018-го), в которых погибли 1775 человек (1969 жертв в 2018-м) и пострадали свыше 19,4 тыс. (почти 20,1 тысяч в 2018-м).

И несмотря на то, что показатель числа ДТП уменьшается, решение вопроса о снижении числа транспортных происшествий остается актуальным. В городах увеличивается количество машин, происходит рост числа населения, в следствие чего возникает необходимость оценить загруженность пешеходных переходов, что приведет к решению проблем на дорогах, а также сокращению числа аварий с участием пешеходов. Развитие компьютерного зрения дает возможность решить поставленную задачу. Хорошо в решении данной задачи зарекомендовали себя искусственные нейронные сети (ИНС).

Целью данной работы является разработка модели ИНС, которая в режиме реального времени обнаруживает и ведет подсчет пешеходов в каждом направлении дорожного узла.

Для достижения данной цели необходимо решить следующие задачи:

1) проанализировать существующие подходы к решению задачи распознавания пешеходов;

2) разработать математическую модель ИНС;

3) проанализировать, собрать и подготовить данные для обучения ИНС;

4) проверить работу ИНС в дорожных узлах в режиме реального времени.

1 МЕТОДЫ ДЕТЕКТИРОВАНИЯ ПЕШЕХОДОВ В РЕЖИМЕ РЕАЛЬНОГО ВРЕМЕНИ

1.1 Этапы детектирования пешеходов

Детектирование пешеходов подразделяется на два основных этапа: в первом происходит выбор региона изображения, который может оказаться «пешеходом», а во втором определяется, выбранный регион – «пешеход» или нет. Данные компоненты называются соответственно «генерация кандидатов» и «классификация». Модификация этих этапов и соответствующих им алгоритмов приводит к созданию новых методов детектирования пешеходов. При этом структура системы остается неизменной.

Обобщенная структура системы детектирования пешеходов (рисунок 1.1) состоит из следующих модулей:

1) предварительная обработка – получение входных данных и их преобразование для последующей обработки;

2) генерация кандидатов – извлечение регионов изображения, которые представляют интерес и передача их классификатору, причем необходимо отсеять столько регионов – «не пешеходов», сколько возможно;

3) классификация – определение принадлежности кандидата к классам «пешеходов» и «не пешеходов»;

4) верификация и уточнение, где верификация подразумевает фильтрацию ложноположительных результатов, а уточнение – дальнейшее уточнение области с пешеходом;

5) трекинг – непрерывное отслеживание перемещения детектированного пешехода для исключения ошибочного детектирования, предсказания положения пешехода в следующий момент времени и так далее;

6) применение – использование результатов работы предыдущих модулей.

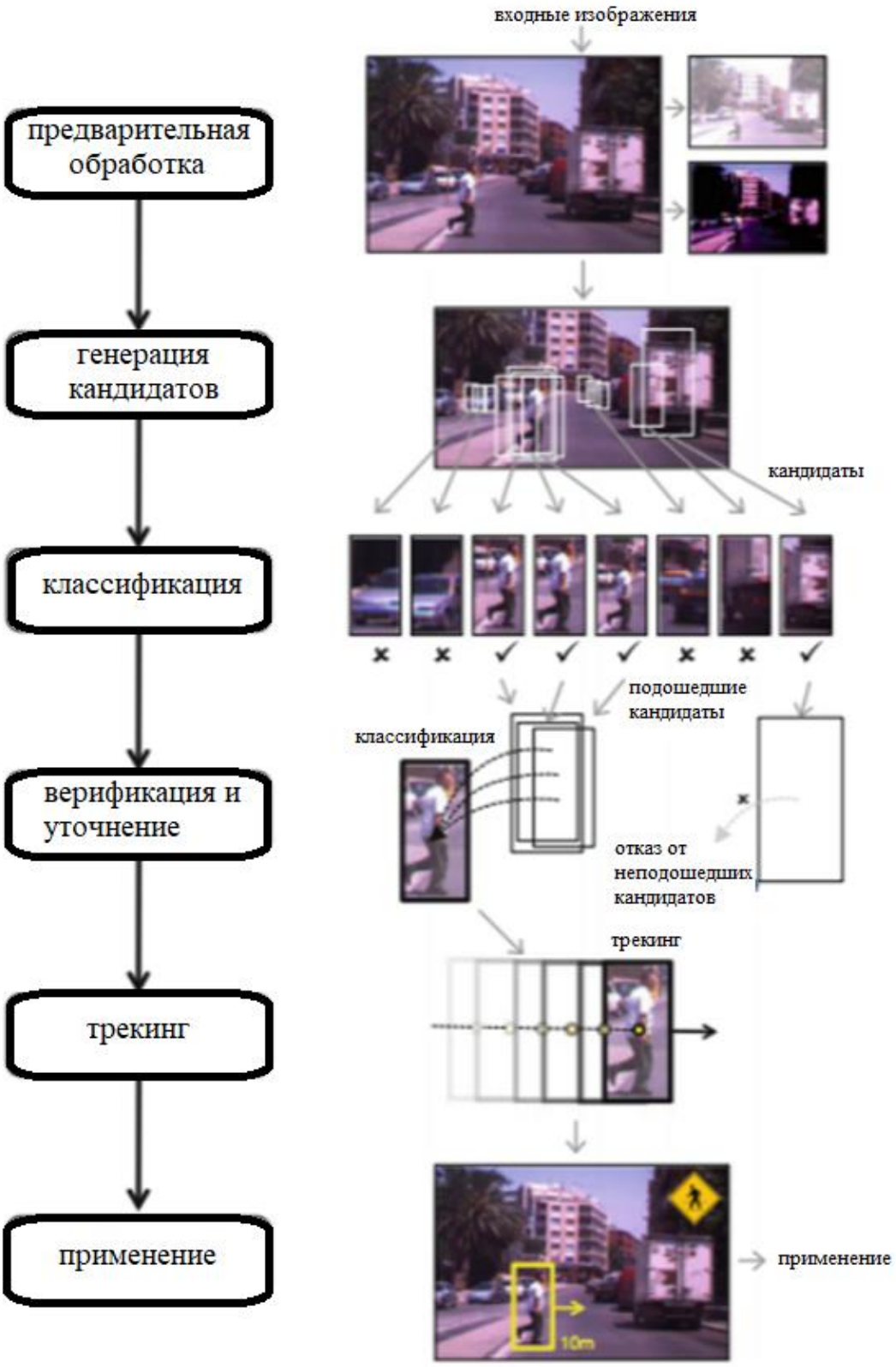

Рисунок 1.1 – Обобщенная структура системы детектирования пешеходов

На каждом из этапов могут возникать различные проблемы. К примеру, на этапе предварительной обработки могут быть получены некорректные входные данные, что приведет за собой ряд других проблем в процессе работы

с ними. На этапах 2–4 могут быть неверно выбраны регионы изображения и классифицированы как «пешеходы». На 5-ом этапе, в процессе отслеживания пешехода, он может периодически «теряться», т. е. в завсисимости от его положения он может быть классифицирован не как «пешеход». На последнем этапе могут возникнуть проблемы из-за некорректных результатов предыдущих модулей.

Рассмотрим такие методы детектирования объектов, как компьютерное зрение и глубокие нейронные сети.

В независимости от метода генерации кандидатов классификация происходит по следующей схеме: выделение признаков, нормализация, классификация [3]. Под выделением признаков подразумевается процесс, включающий ряд процедур преобразования видеоданных. Нормализация – это приведение входных данных к нормальному состоянию, к примеру, приведение их к определенному интервалу ($[-1; 1]$ и др.). Также в нее входят процедуры, которые устраняют основные типы искажений в исходных данных. Классификация – это определение принадлежности кандидата к какому-либо классу. Далее рассмотрим подробнее каждый из методов детектирования.

1.2 Компьютерное зрение

Машинное (компьютерное, техническое) зрение – это научное направление в области искусственного интеллекта, в частности робототехники, и связанные с ним технологии получения изображений объектов реального мира, их обработки, и использования полученных данных для решения разного рода прикладных задач без участия человека.

Библиотека OpenCV – это популярное программное обеспечение для создания систем распознавания образов – систем компьютерного зрения.

OpenCV (Open Source Computer Vision Library) – библиотека с открытым исходным кодом для обработки изображений и численных алгоритмов общего назначения, алгоритмов компьютерного зрения.

Существует множество методов распознавания объектов на изображении. Ниже рассмотрим некоторые из них.

1.2.1 Метод вычитания фона

Одним из наиболее распространенных методов выделения областей движения на видео является метод вычитания фона (background subtraction) [4].

Алгоритм формирует фоновую модель, усредняя заданное число кадров видеопоследовательности, затем выполняются попиксельное вычитание между текущим кадром и фоновой моделью, содержащей статическую часть сцены, и сравнение. Таким образом на черном фоне белым цветом выделяются движущиеся объекты. Для решения конкретной задачи можно фильтровать их по размеру, задавая пороговые значения параметров: высота и ширина, при которых объект будет считаться целевым.

Сложность этого метода состоит в том, что постоянно приходится учитывать параметры съемки, освещенность кадров, дрожание камеры.

Для моделирования алгоритмов вычитания фона используется реализованный в библиотеке OpenCV метод gaussian mixture model for background subtraction [5]. В нем присутствуют операции по шумоподавлению, усилению яркости найденных объектов, а также сам алгоритм вычитания фона. Результаты работы алгоритма представлены на рисунке 1.2. Когда автомобиль неподвижен мы получаем, что вычитание фона дает нулевой результат, т. е. автомобиль не определяется, при передвижении автомобиля, он определяется, но весьма неточно (средний кадр).

Другим примером является работа алгоритма на видео с локомотивом (рисунок 1.3). При передвижении локомотива, алгоритм сразу же четко его находит (средний кадр).

Рисунок 1.2 – Работа алгоритма вычитания фона
на видеопоследовательности с автомобилем

Локомотив довольно четко определяется на всех кадрах видео. Однако также на всех кадрах определяется тень от локомотива (последний кадр). Стоит отметить, что тень, найденная алгоритмом не видна человеческим глазом.

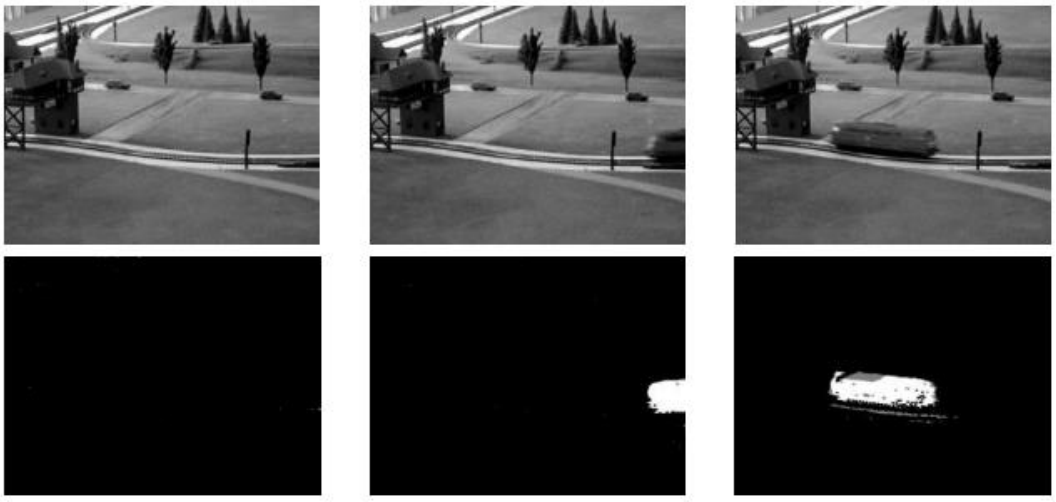

Рисунок 1.3 – Работа алгоритма вычитания фона
на видеопоследовательности с локомотивом

Тень, отбрасываемая объектом, может создать серьезные проблемы в обнаружении самого объекта при работе алгоритма в реальных условиях. На открытом пространстве железнодорожного переезда может возникать множество теней, которые, несомненно, влияют на распознавание объекта. Устранение таких помех решается с помощью дополнительных алгоритмов

или использования дополнительного аппаратного обеспечения. Также к недостаткам этого метода можно отнести то, что если объект на видеоизображении будет неподвижным, то он не будет распознан.

1.2.2 HOG дескриптор

Гистограмма направленных градиентов (Histogram of Oriented Gradients, HOG) – дескриптор, описывающий изменение яркости на изображении яркость и саму структуру этих перепадов, за принятие решений отвечает линейный классификатор Support vector machines (SVM) [6].

На исходном изображении (рисунок 1.4) форма и вид объектов описываются направлением краев или распределением градиентов яркости (рисунок 1.5), в этом заключается основная идея в алгоритме формирования HOG дескриптора.

Рисунок 1.4 – Исходное изображение

Чтобы реализовать такой дескриптор изображение делится на связные области (ячейки) (рисунок 1.6), и ведется подсчет направлений градиентов для каждой ячейки или направлений краев пикселей внутри.

Рисунок 1.5 – Приближенные значения градиентов вдоль осей X и Y, матрицы направлений и силы градиента

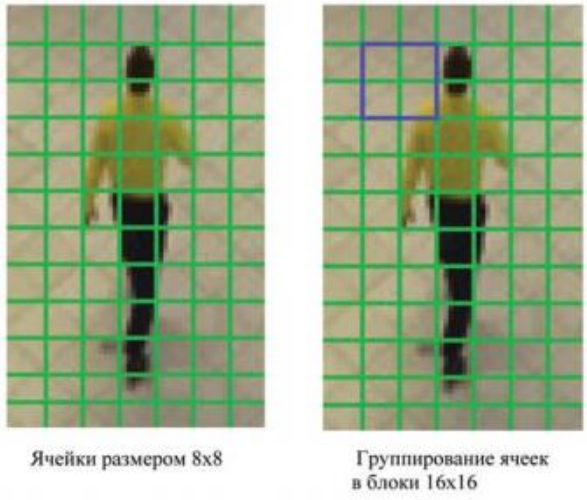

Рисунок 1.6 – Группирование ячеек в более крупные связные блоки

Комбинация гистограмм (рисунок 1.7) называется дескриптором. Для увеличения точности производится нормализация по контрасту для локальных гистограмм. Сначала считается мера интенсивности (блок) на большей облас-

ти изображения, после чего получившееся значение применяется для нормализации. Нормализация понижает восприимчивость к изменениям освещения у дескрипторов.

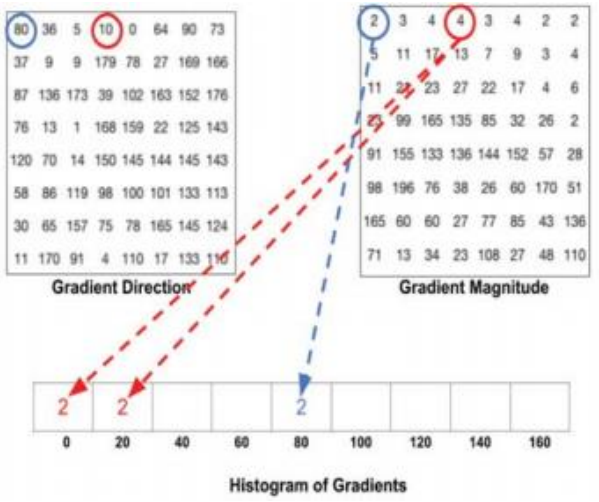

Рисунок 1.7 – Расчет гистограммы ячеек

Для обучения алгоритма распознаванию людей используется машинное обучение, а именно метод опорных векторов (Linear SVM). Обученный линейный SVM-классификатор представим в виде вектора коэффициентов уравнения разделяющей гиперплоскости в пространстве признаков.

Метод опорных векторов заключается в переводе полученных векторов в пространство размерности выше и поиск гиперплоскости, которая разделит классы, с максимально возможным зазором в этом пространстве. Разделяющая гиперплоскость – это гиперплоскость, максимизирующая расстояние до наших параллельных гиперплоскостей. Средняя ошибка классификатора зависит от расстояния или разницы между параллельными гиперплоскостями, чем больше один из этих показателей, тем меньше будет средняя ошибка классификатора. Классификация заключается в необходимости нахождения того, с какой стороны относительно гиперплоскости находится вектор, который нужно классифицировать.

Формально этот метод можно описать следующим образом. Мы полагаем, что точки имеют вид (формула 1.1):

$$\{(x_1, c_1), \dots, (x_n, c_n)\}, \tag{1.1}$$

где x_i – это p-мерный вещественный вектор, обычно нормализованный значениями $[0,1]$ или $[-1,1]$;

c_i – показатель, принимающий значение 1 или -1, в зависимости от того, к какому классу принадлежит точка x_i.

В случае отсутствия нормализации точек, точка с большим отклонением слишком сильно повлияет на классификатор. Рассмотрим это как учебную коллекцию, где каждому элементу уже задан класс, к которому он принадлежит. Необходимо, чтобы алгоритм метода опорных векторов классифицировал их таким же образом. Для этого строится разделяющую гиперплоскость, которая имеет вид (формула 1.2):

$$wx - b = 0, \tag{1.2}$$

где w – перпендикуляр к разделяющей гиперплоскости;

b – скалярный порог.

Параметр $\frac{b}{\|w\|}$ равен по модулю расстоянию от гиперплоскости до начала координат. В случае равенства параметра b нулю, гиперплоскость проходит через начало координат, что ограничивает решение.

В связи с тем, что нам нужно найти оптимальное разделение, нас интересуют опорные вектора и гиперплоскости, параллельные оптимальной и ближайшие к опорным векторам двух классов. Эти параллельные гиперплоскости описываются следующими уравнениями (с точностью до нормировки) (формула 1.3).

$$\begin{bmatrix} wx - b = 1, \\ wx - b = -1,' \end{bmatrix} \tag{1.3}$$

где w – перпендикуляр к разделяющей гиперплоскости;

x – это p-мерный вещественный вектор, обычно нормализованный значениями $[0,1]$ или $[-1,1]$;

b – скалярный порог.

В случае линейной разделимости обучающей выборки можно выбрать гиперплоскости так, чтобы между ними не было ни одной точки обучающей выборки, затем максимизировать расстояние между гиперплоскостями. Ширина полосы между ними находится из соображений геометрии, она равна $\frac{2}{\|w\|}$, таким образом, наша задача минимизировать $\|w\|$. Чтобы исключить все точки из полосы, мы должны убедиться для всех i, что (формула 1.4):

$$\begin{bmatrix} wx_i - b \geq 1, & c_i = 1, \\ wx_i - b \geq -1, & c_i = -1,' \end{bmatrix} \quad (1.4)$$

где w – перпендикуляр к разделяющей гиперплоскости;

$x -$ это p-мерный вещественный вектор, обычно нормализованный значениями $[0,1]$ или $[-1,1]$;

$c_i -$ принимает значение 1 или -1, в зависимости от того, к какому классу принадлежит точка x_i;

$b -$ скалярный порог.

Результат классификации полученных дескрипторов при помощи линейного классификатора (SVM) представлен на рисунке 1.8.

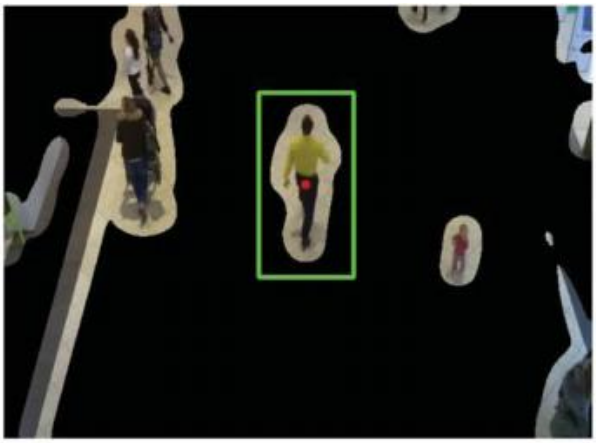

Рисунок 1.8 – Результат решения классификатора

Несмотря на преимущества метода HOG, который позволяет детектировать непосредственно людей, он не обеспечивает должного качества распознавания в условиях недостаточной освещенности, расположения камеры под углом и т. д. Помимо этого алгоритм довольно ресурсоемок.

1.2.3 Поиск контуров

Алгоритм поиска контура подразделяется на три этапа: преобразование изображения в изображение с усиливающими границами, поиск контуров среди выделенных границ и последующая обработка контуров в соответствии с условиями задачи.

Поиск контуров является наиболее трудоемкой с вычислительной точки зрения задачей. Элементом контура называется граница между двумя соседними пикселями. Стоимостью элемента контура называется разница между максимальной яркостью пикселей изображения и разницей между яркостями пикселей элемента контура. Контур состоит из связанного множества элементов контура с минимальной стоимостью [7]. Если представить изображение в виде графа, вершинами которого являются пиксели, а ребра связывают смежные вершины и имеют вес, равный стоимости элемента контура, то задача поиска контура сводится к задаче поиска минимального пути в этом графе.

Для моделирования алгоритмов выделения контуров используем реализацию алгоритма в библиотеке OpenCV, для выделения границ используем детектор Кэнни. Работа данного детектора включает в себя следующие шаги:

– убрать шум и лишние детали из изображения;

– рассчитать градиент изображения;

– сделать края тонкими;

– связать края контура.

Результат работы алгоритма после проведения моделирования для кадров нескольких видеопоследовательностей представлен на рисунке 1.9.

Из приведенных кадров видно, что в целом контуры автомобиля определяются. При этом сам автомобиль распадается на несколько контуров, часть из которых имеет малую площадь и на рисунке не отображена. При этом определяется также большое число контуров других элементов. Причем эти

контуры изменяются от кадра к кадру, что затрудняет определение постоянных объектов в видеопоследовательности по их контурам.

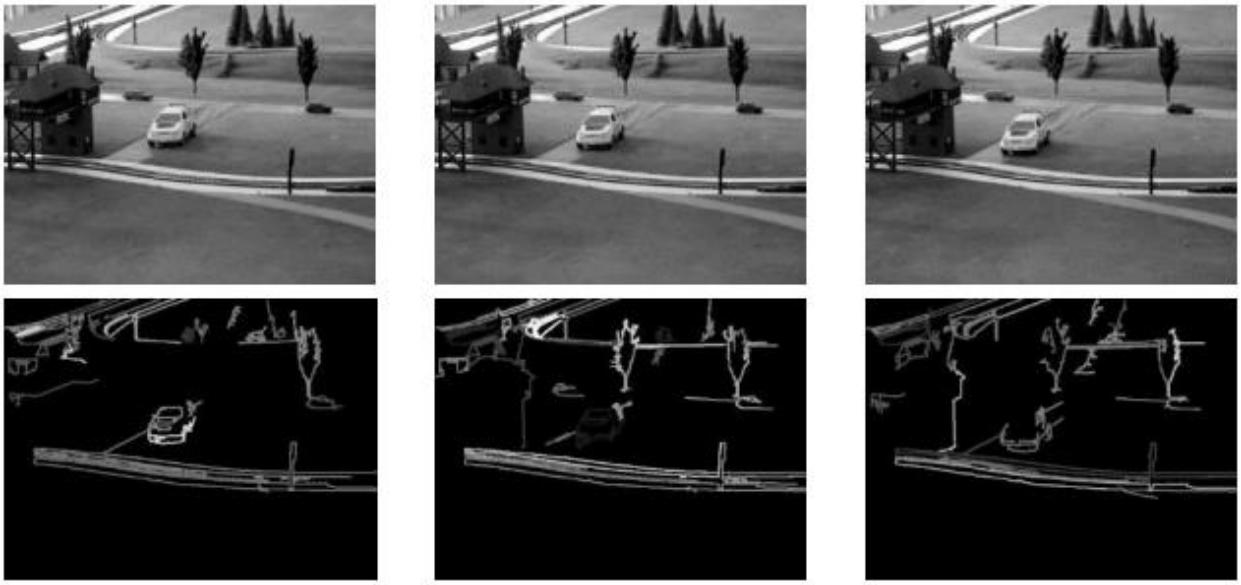

Рисунок 1.9 – Моделирование алгоритма определения контуров на видеопоследовательности с автомобилем

Характерные результаты работы алгоритма определения контуров для видеопоследовательности с проезжающим локомотивом представлены на рисунке 1.10.

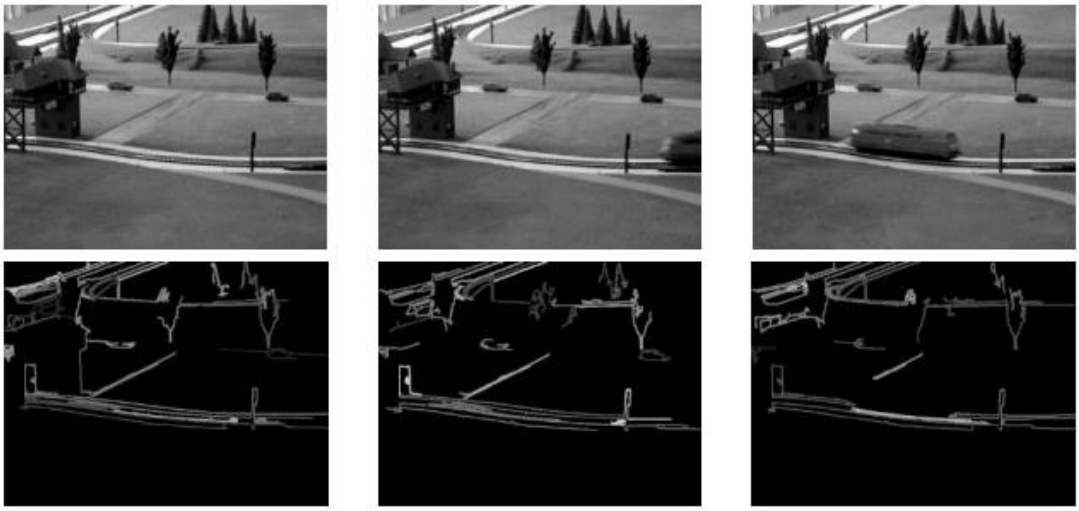

Рисунок 1.10 – Моделирование алгоритма определения контуров на видеопоследовательности с локомотивом

Данный пример представляет собой самый сложный случай для алгоритма определения контуров. Как видно из приведенных данных, контуры движущегося локомотива практически не определяются. Анализ описанной

ситуации показал, что проблема состоит не в алгоритме поиска контуров, а в детектировании границ.

1.3 Глубокие нейронные сети

Среди нейронных сетей, применяемых при решении задач распознавания образов, особой популярностью пользуются CNN, поскольку они были разработаны специально для решения задач, связанных с обработкой изображений [8]. Свёрточная нейронная сеть (CNN) – это искусственная нейронная сеть, имеющая особую архитектуру, нацеленную на эффективное распознавание изображений. Входит в состав технологий глубинного обучения (deep learning) [9]. CNN используются в задачах классификации, сегментации, распознавания и др. Рассмотрим наиболее популярные сверточные нейронные сети.

1.3.1 Алгоритм R-CNN

R-CNN – алгоритм детектирования объектов на изображении, при помощи использования простых «сильных» и сложных «слабых» классификаторов, с заданной точностью распознавания [10].

Для решения задачи распознавания объектов (распознавания пешеходов в нашем случае) R-CNN является, можно сказать, первой моделью. Система обнаружения объектов R-CNN использует алгоритм Selective Search или взаимозаменяемые (Selective Search – это алгоритм выборочного поиска), работа которого подразделяется на 3 этапа (рисунок 1.11):

Первый этап – генерация около 2000 областей, которые с высокой вероятностью содержат какой-либо объект, отличный от фона [11].

После того, как предлагаемые области сгенерированы, каждая область обрезается до размера 227×227 пикселей и далее подается на вход CNN, которая пропускает ее через 5 сверточных слоев, 2 полносвязных слоя и выдает в виде фиксированного 4096-размерного вектора признаков.

Финальный этап – вектор признаков подается на вход набору линейных персептронов (персептрон – математическая или компьютерная модель восприятия информации мозгом) на основе метода опорных векторов, обученных для каждого класса, и выполняется классификация. Этот вектор также подается на вход регрессии для вычисления максимально точных координат ограничивающей рамки. Далее выполняется алгоритм подавления немаксимумов, который отклоняет регион, если он в значительной степени совпадает с другим регионом, имеющим более высокую оценку.

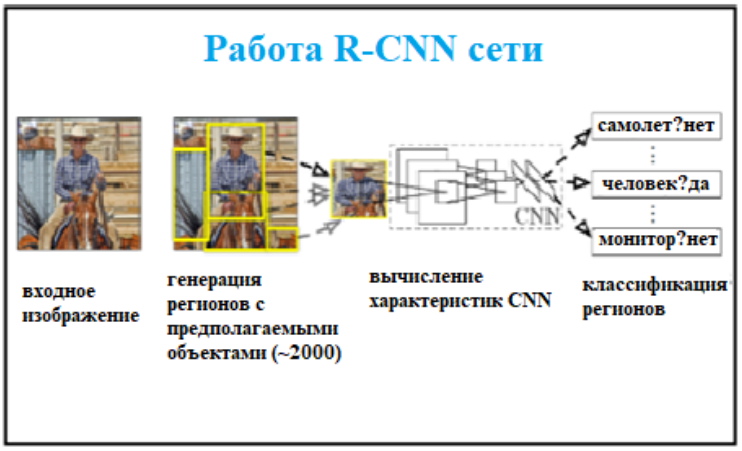

Рисунок 1.11 – Работа R-CNN сети

У R-CNN имеются некоторые недостатки, самый существенный из них заключается в том, что необходимо порядка 2–3 дней работы GPU для обучения алгоритма. Также при обработке нужно порядка 47 секунд на изображение, что довольно-таки долго.

Таким образом, главное слабое место состоит в том, что каждое отдельное окно предполагаемого объекта требуется пропустить через CNN.

1.3.2 Алгоритм Fast R-CNN

Fast R-CNN – модификация алгоритма R-CNN, основанная на использовании нейронных сетей для построения ROI. ROI (Region of Interest, регион интересов) – интересующие области изображений, «зоны интереса».

Была предложена в 2015 году с улучшенными характеристиками по сравнению с R-CNN. Возросла точность обнаружения mAP с 62% до 66% на

VOC 2012, увеличена скорость обучения в 9 раз, а также скорость обработки в 213 раз.

Работа Fast R-CNN (рисунок 1.12) заключается в том, что на вход нейросети подается изображение и набор окон с предполагаемыми объектами [12]. Далее несколько сверточных слоев и слоев выбора максимального элемента (max pooling) производят карту признаков (feature maps). Следующий слой RoI выделяет один вектор признаков из этой карты для каждого предположения объекта, который в итоге пропускается через серию полносвязных слоев и передается в два выходных слоя:

– слой softmax, который по каждому классу производит оценку вероятности;

– слой регрессора обрамляющих окон (bounding box regressor), возвращающий уточненные позиции обрамляющего окна.

Fast R-CNN имеет хорошую скорость обработки изображения (таблица 1.1), но он все еще опирается на медленные алгоритмы, генерирующие окна с предполагаемыми объектами. Selective Search обычно требует порядка 1–2 секунд на изображение, тем самым ограничивая частоту обработки значением в 0.5 FPS (кадровая частота) и сводя на нет все дальнейшие улучшения скорости работы сверточной нейросети.

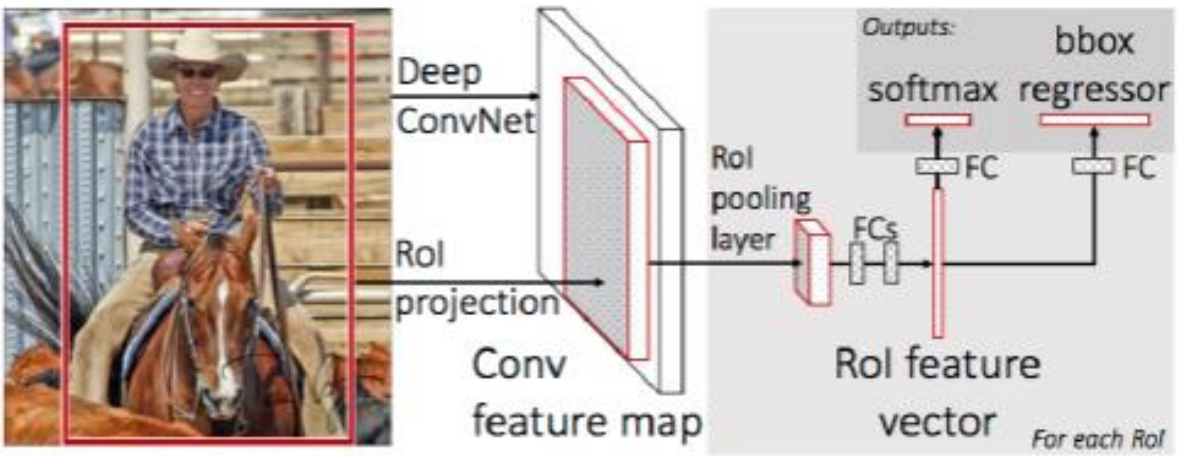

Рисунок 1.12 – Схема Fast R-CNN

Таблица 1.1 – Результаты Fast R-CNN в сравнении с R-CNN

	R-CNN	Fast R-CNN
Время обучения:	84 часа	9.5 часов
Ускорение времени обучения	1x	8.8x
Средняя точность предсказания (mAP (VOC 2007))	66.0%	66.9%
Время обработки одного изображения сетью с помощью алгоритма Selective Search	50 секунд	2 секунды
Ускорение времени обработки одного изображения	1x	25x

1.3.3 Алгоритм Faster R-CNN

Faster R-CNN – главное отличие состоит в том, что вместо Selective Search алгоритма для выбора регионов использует нейронную сеть для их «заучивания».

Faster R-CNN (рисунок 1.13) является улучшенной версией Fast R-CNN. В данном алгоритме решены проблемы долгого и не очень точного поиска объектов на изображении [13]. Средняя точность mAP увеличилась с 66% до 70,4% на VOC 2012, а также была увеличена скорость работы (таблица 1.2). На первом этапе Faster R-CNN использует собственную глубокую полносвязную нейросеть RPN (Region Proposal Networks), а на втором алгоритм Fast R-CNN.

Идея RPN основана на понимании того, что Fast R-CNN требует предположения по окнам с объектами только после первых сверточных слоев, а карты признаков, которые получаются после этих первых слоев, могут быть использованы для генерации окон с предполагаемыми объектами. Вследствие чего сокращается большее количество вычислений, если внести этап поиска регионов в нейросеть.

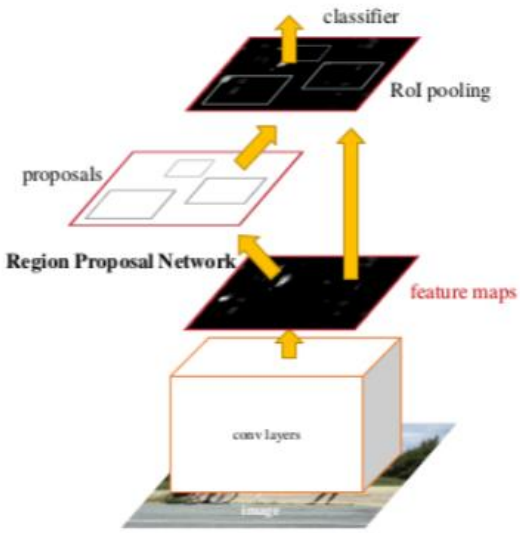

Рисунок 1.13 – Схема Faster R-CNN

Таблица 1.2 – Сравнительные характеристики

	R-CNN	Fast R-CNN	Faster R-CNN
Время обработки одного изображения сетью	50 секунд	2 секунды	0.2 секунды
Ускорение времени обработки одного изображения	1x	25x	250x
Средняя точность предсказания (mAP (VOC 2007))	66.0%	66.9%	66.9%

RPN строится путем добавления 2 новых сверточных слоев поверх общих сверточных уровней Fast R-CNN. Так как большая часть вычислений в RPN используется совместно с сетью обнаружения, только стоимость двух дополнительных уровней влияет на скорость всего алгоритма Faster R-CNN.

1.3.4 Алгоритм YOLO

Главная особенность YOLO заключается в том, что CNN применяется один раз ко всему изображению сразу, в отличие от большинства других систем, которые применяют CNN несколько раз к разным регионам изображения [14].

На выход CNN отдает тензор, которого достаточно для построения границ обнаруженных объектов. Его размер представим в виде формулы 1.5:

$$S * S * (B * 5 + K), \quad (1.5)$$

где S – размерность сетки;

B – количество границ, используемых в оценке для каждой ячейки;

K – количество классов, которые нейронная сеть способна распознавать.

Архитектура YOLO представлена на рисунке 1.14.

Рисунок 1.14 – Архитектура YOLO

Система детектирования работает следующим образом [15].

1. Изображение разбивается на $S * S$ областей. Каждой ячейке области соответствует вектор размерностью $(B * 5 + K)$, где число 5 определяет количество показателей каждой границы. Используемые показатели: x, y, w, h, k, где (x, y) – координаты центра объекта относительно ячейки (принимают значения от 0 до 1 по отношению к размеру ячейки), w, h – ширина и высота объекта соответственно (имеют значения от 0 до 1 по отношению к ширине и высоте исходного изображения соответственно), k –вероятность того, что граница верно определена. Первые $B * 5$ значений вектора, соответствующей ячейки, характеризуют именно эти параметры. Оставшиеся K значений, в этом векторе показывают вероятности того, что центр объекта находится в этой ячейке. На данный момент эти вероятности не привязаны к границам, для нахождения вероятностей классов для каждой из границ необходимо умножить характеристику границы k на эти K значений вектора.

2. В результате получаем границ $S * S * B$ с вероятностями классов. Для того чтобы получить итоговое распознавание необходимо выполнить следующее:

1) фиксируется какой-либо класс, например «пешеход», для которого имеется вектор со значением вероятности класса «пешеход» с каждой из $S * S * B$ границ;

2) обнуляем значения для тех границ, у которых значение меньше порогового (задается заранее);

3) сортируем вектор по убыванию;

4) применяем алгоритм NMS (Non maximal suppression). Работает он по следующему принципу: на входе подается вектор из $S * S * B$ вероятностей по классу «пешеход» для всех границ. Выбирается максимальное значение, так как вектор отсортирован, то этот элемент находится на первой позиции, и затем эта граница сравнивается с границами, расположенными правее, у которых вероятность по классу больше нуля. Сравнение происходит по площади пересечения: если площадь пересечения больше 0.5, то для границы с меньшей вероятностью эта вероятность обнуляется. По этому принципу сравниваем остальные границы. А затем фиксируем следующую границу с ненулевой вероятностью и проводим аналогичные операции сравнения. Таким образом, рассматриваются все границы для класса «пешеход»;

5) далее берется следующий класс, и проводятся аналогичные операции сравнения и обнуления. После подобной процедуры получаются разряженные вектора с вероятностями по каждому классу для каждой найденной границы.

Остается только решить какие границы необходимо нанести на исходное изображение. Метод отбора довольно прост: рассматривается каждая граница, берется максимальное значение вероятности по классам и, если оно больше нуля, то граница наносится на исходное изображение, иначе пропускается и рассматривается следующая.

YOLO в 1000 раз быстрее чем R-CNN и в 100 раз быстрее чем Fast R-CNN. Количество правильно определенных объектов на выборке с помощью YOLO составляет 65,5%, а у R-CNN около 71,6%.

1.3.5 Алгоритм YOLOv3-tiny

YOLOv3-tiny – обрезанная версия архитектуры YOLOv3. В ней всего 2 выходных слоя. Она хуже предсказывает мелкие объекты и предназначена для работы с небольшими наборами данных. Но, благодаря урезанному строению, веса сети занимают небольшой объем памяти (~35 Мб) и она выдает более высокий FPS. Следовательно, для использования на мобильном устройстве, данная архитектура является предпочтительной.

1.3.6 Алгоритм YOLOv3

YOLOv3 – улучшенная версия архитектуры YOLO, состоящая из 106-ти свёрточных слоев. Она лучше детектирует небольшие объекты по сравнению с её предшественниками. Главная особенность YOLOv3 заключается в том, что на выходе есть 3 слоя, рассчитанных на обнаружение объектов разного размера [16]. Было произведено сравнение YOLOv3 с другими детекторами, которое представлено на рисунке 1.15.

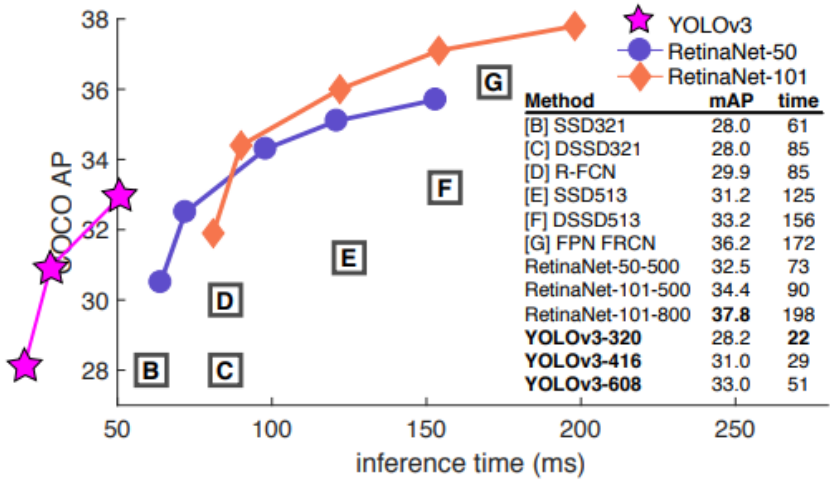

Рисунок 1.15 – Сравнение YOLOv3 с другими детекторами

1.3.7 Алгоритм SSD

SSD – по принципу похожа на YOLO, но в качестве сети для извлечения признаков использует VGG16. VGG16 – это нейронная сеть глубокого обучения, разработанная Visual Geometry Group для распознавания объектов на изображениях, состоящая из 16 слоёв [17].

SSD был выпущен в конце ноября 2016 года и достиг новых рекордов по производительности и точности для задач обнаружения объектов, набрав более 74% mAP на 59 кадрах в секунду на стандартных наборах данных, таких как PascalVOC и COCO. Чтобы лучше понять SSD, нужно начать с объяснения, откуда происходит название этой архитектуры.

1. Single Shot означает, что решаются задачи локализации и классификации объектов за один проход нейросети.

2. MultiBox – это метод для регрессии ограничивающего прямоугольника.

3. Detector является сетью (детектор объектов), который также классифицирует обнаруженные объекты.

Архитектура SSD (рисунок 1.16) основана на старой архитектуре VGG-16 (рисунок 1.17), исключая полностью подключенные слои. Причина использовать VGG-16 как базовую сеть связана с его высокой производительностью в задачах классификации изображений высокого качества и его популярностью для задач, где передача обучения помогает в улучшении результатов. Вместо исходных VGG с полностью подключенными слоями, был добавлен набор вспомогательных сверточных слоев, что позволяет извлекать объекты в нескольких масштабах и постепенно уменьшать размер входных данных для каждого последующего слоя.

SSD обеспечивает худшую производительность для небольших объектов, так как они могут не отображаться на всех картах объектов. Увеличение разрешения входного изображения облегчает эту проблему, но не решает ее полностью.

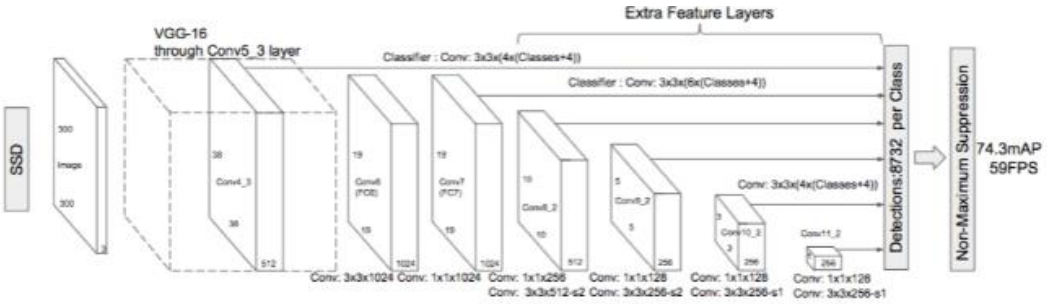

Рисунок 1.16 – Архитектура SSD

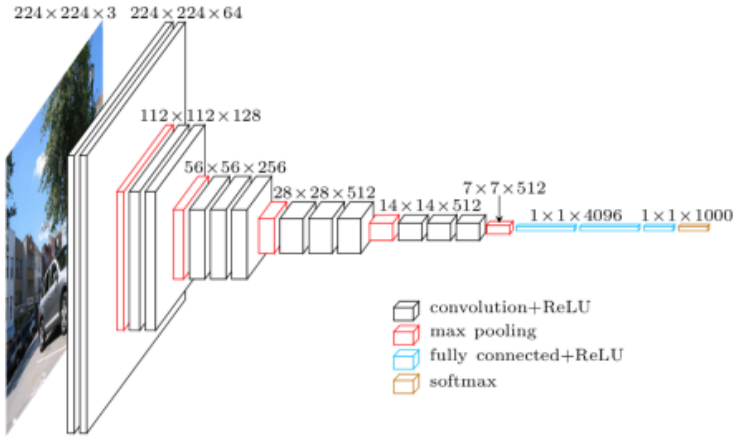

Рисунок 1.17 – Архитектура VGG

1.3.8 Алгоритм FPN

FPN (англ. Feature Pyramid Networks) – еще одна разновидность сети типа Single Shot Detector, из-за особенности извлечения признаков лучше, чем SSD распознает мелкие объекты. Назначение Feature Pyramids состоит в улучшении качества детектирования объектов с учётом большого диапазона их возможных размеров [18].

Карты признаков в FPN, извлечённые последовательными слоями CNN с уменьшающейся размерностью, рассматриваются как своего рода иерархическая «пирамида», называемая: путь снизу-вверх. Карты признаков и нижних, и верхних уровней пирамиды имеют свои достоинства и недостатки: у первых высокое разрешение, но низкая семантическая, обобщающая, способность; у вторых – наоборот (рисунок 1.18).

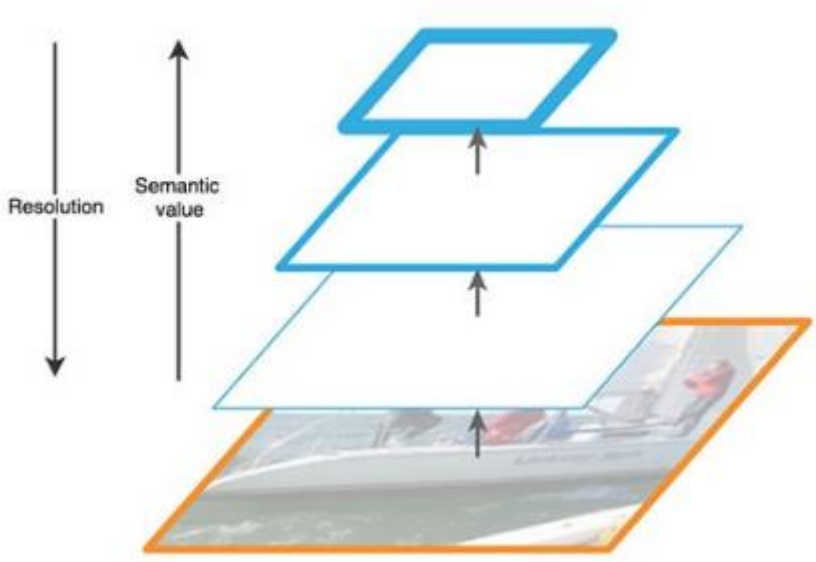

Рисунок 1.18 – Карты признаков

Архитектура FPN (рисунок 1.19) путем добавления пути сверху-вниз и боковых соединений позволяет объединить достоинства верхних и нижних слоев. Для этого карта каждого вышележащего слоя увеличивается до размера нижележащего и их содержимое поэлементно складывается. В итоговых предсказаниях используются результирующие карты всех уровней.

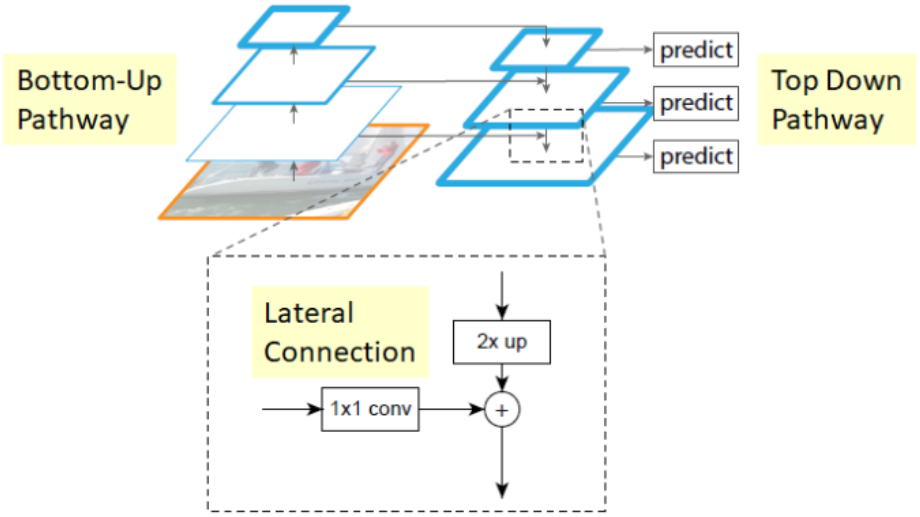

Рисунок 1.19 – Архитектура FPN

Чтобы протестировать то, как FPN справляется с задачей сегментации было произведено сравнение с другими подходами на стандартном наборе данных COCO (рисунок 1.20).

method	backbone	competition	image pyramid	test-dev					test-std				
				$AP_{@.5}$	AP	AP_s	AP_m	AP_l	$AP_{@.5}$	AP	AP_s	AP_m	AP_l
ours, Faster R-CNN on **FPN**	ResNet-101	-		**59.1**	**36.2**	**18.2**	**39.0**	48.2	**58.5**	**35.8**	**17.5**	**38.7**	47.8
*Competition-winning **single-model** results follow:*													
G-RMI†	Inception-ResNet	2016		-	34.7	-	-	-	-	-	-	-	-
AttractioNet‡ [10]	VGG16 + Wide ResNet§	2016	✓	53.4	35.7	15.6	38.0	**52.7**	52.9	35.3	14.7	37.6	**51.9**
Faster R-CNN +++ [16]	ResNet-101	2015	✓	55.7	34.9	15.6	38.7	50.9	-	-	-	-	-
Multipath [40] (on minival)	VGG-16	2015		49.6	31.5	-	-	-	-	-	-	-	-
ION‡ [2]	VGG-16	2015		53.4	31.2	12.8	32.9	45.2	52.9	30.7	11.8	32.8	44.8

Рисунок 1.20 – Сравнение FPN с другими подходами

1.4 Выводы по главе 1

Задача распознавания пешеходов в режиме реального времени требует высоких затрат как вычислительных ресурсов, так и дополнительных объемов памяти и времени выполнения. Необходимо обрабатывать большие потоки данных и хранить данные результатов обучения при использовании алгоритмов классификации.

В связи с этим, необходимо выбрать наиболее подходящий метод для решения данной задачи. Метод вычитания фона не способен детектировать неподвижные объекты, т.е. если пешеход будет стоять на месте, он не будет распознан. Метод HOG довольно ресурсоемок. А алгоритм поиска контуров детектирует множество лишних объектов. Помимо этого, данные методы не очень точно распознают объекты, особенно в условиях плохой освещенности, попадания теней на объекты и пр. Поэтому для решения поставленной задачи целесообразно использовать сверточные нейронные сети, которые точнее детектируют объекты и обладают большим числом достоинств по сравнению с представленными методами компьютерного зрения.

Каждая из описанных архитектур CNN обладает своими достоинствами и недостатками. И больше всего для задачи распознавания пешеходов в режиме реального времени подходит YOLOv3. Данная архитектура отлично справляется с детектированием объектов в режиме реального времени в отличие от других архитектур, таких как R-CNN и прочих, она довольно точно определяет объекты, в том числе и небольших пешеходов, в отличие от SSD. Также очевидны преимущества подхода YOLOv3: для нахождения объектов

на изображении и их классификации необходима одна нейронная сеть, в отличие от сетей подобных R-CNN.

В качестве фреймворка будет использоваться DarkNet. Это фреймворк с открытым исходным кодом, написанный на языке Си с использованием программно-аппаратной архитектуры параллельных вычислений CUDA. Он довольно быстрый, простой в использовании и вполне подходит для нашей задачи (распознавание пешеходов).

В качестве среды разработки будет использован PyCharm (интегрированная среда разработки для языка программирования Python). Для разметки изображений выбрана платформа Sypervisely.

2 ДЕТЕКЦИЯ ПЕШЕХОДОВ С ПОМОЩЬЮ НЕЙРОННОЙ СЕТИ YOLOV3

2.1 Архитектура нейронной сети

YOLOv3 состоит из магистральной сети, называемой DarkNet-53, и в совокупности содержит 106 слоев – 75 сверточных слоев, а также остаточные блоки, слои с повышенной дискретизацией и пропускаемые соединения. Архитектура сети представлена на рисунке 2.1.

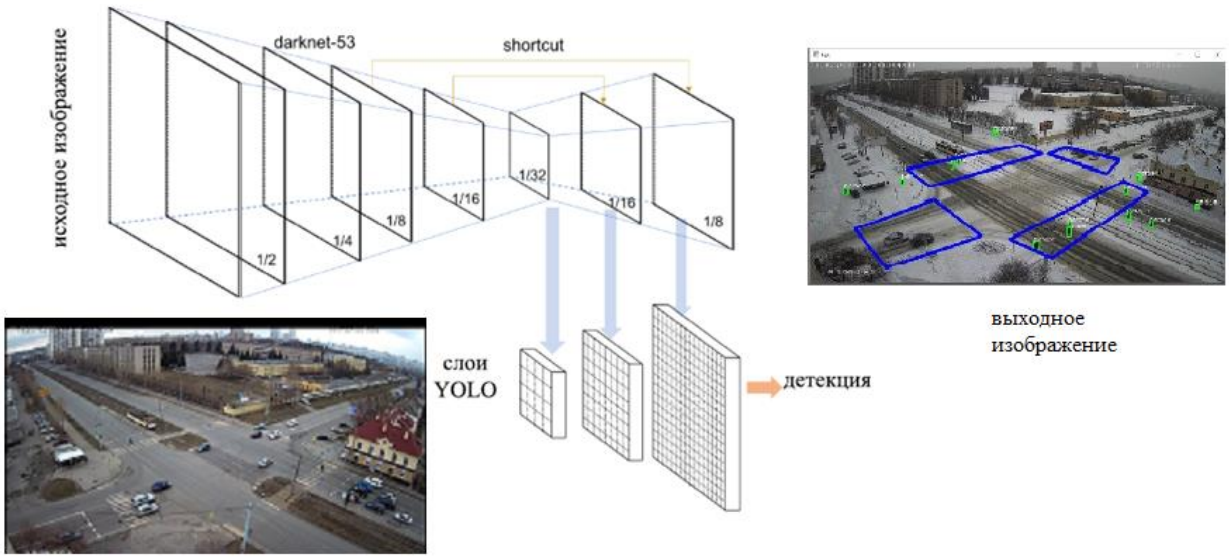

Рисунок 2.1 – Архитектура YOLOv3

Магистральная сеть DarkNet-53 включает в себя 53 сверточных слоя. Сеть принимает остаточные блоки как базовые компоненты. В традиционных нейронных сетях каждый слой поступает в следующий слой. В сети с остаточными блоками каждый слой поступает в следующий слой и непосредственно в слои на расстоянии около 2–3 прыжков [19].

Каждый остаточный блок состоит из пары сверточных слоев 3×3 и 1×1 с shortcut соединением, необходимым для предотвращения затухания градиента (вектора, указывающего, в какую сторону будет расти функция). Слой shortcut схож с остаточным слоем (residual) в нейронных сетях. Он объединяет карты признаков с предыдущих слоев с картами признаков текущего слоя.

На первом слое YOLO разрешение сетки составляет 1/32 входного изображения, итоговое разрешение на последнем слое составляет 1/8 входного

изображения. Это позволяет обнаруживать как мелкие, так и крупные объекты. Между слоями YOLO также находятся слои свертки, route слои, объединяющие массив предыдущих слоев вместе и upsampling слои, предназначенные для увеличения изображения.

2.2 Подготовка данных

Для обучения нейросети требовалось собрать базу данных, состоящую из пешеходов. За внимание были взяты 7 камер различных дорожных узлов. С них было собрано около 7000 изображений, на которых размечен каждый из пешеходов. В сумме эти изображения содержат 105 835 объектов. Далее, путем аугментации (зеркального отображения изображения, увеличения и пр.), полученный набор данных был увеличен в 12 раз. В итоге получилась база данных, состоящая из 84 000 изображений. Среди них 16 800 изображения представляют собой валидационную выборку, оставшиеся – тренировочную. Изображение должно быть в формате JPEG или PNG. На рисунке 2.2 показан пример размеченного изображения.

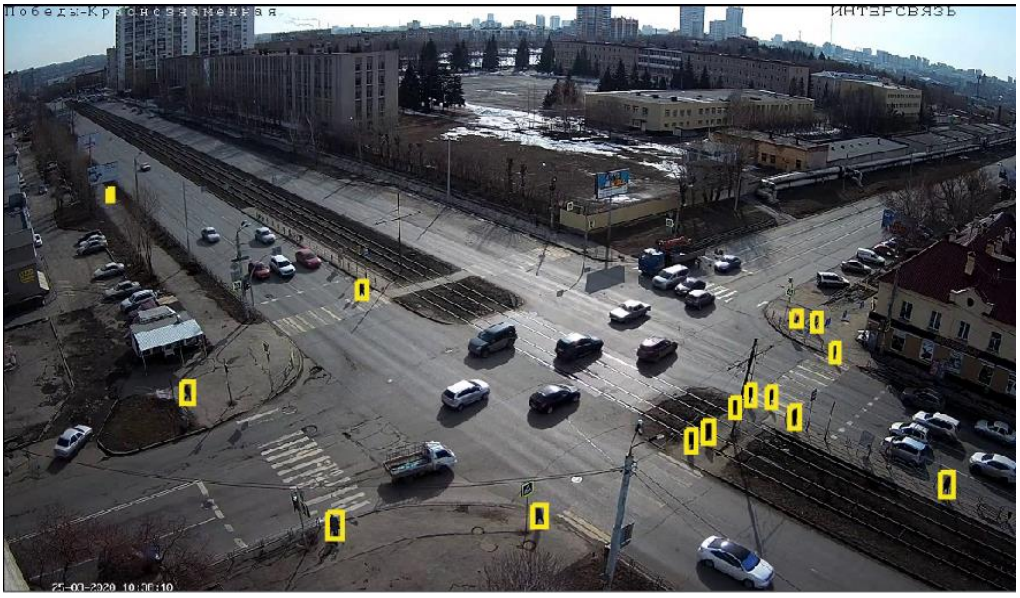

Рисунок 2.2 – Пример размеченного изображения

Каждому изображению соответствует текстовый документ, состоящий из нескольких строк (в зависимости от количества объектов на изображении)

и содержащий в себе 4 показателя: координаты левого верхнего угла ограничивающей рамки вокруг пешехода (x_1, y_1) и координаты правого нижнего угла ограничивающей рамки (x_2, y_2).

Для дальнейшей работы с данными, каждый текстовый документ был приведен к такому виду (рисунок 2.3):

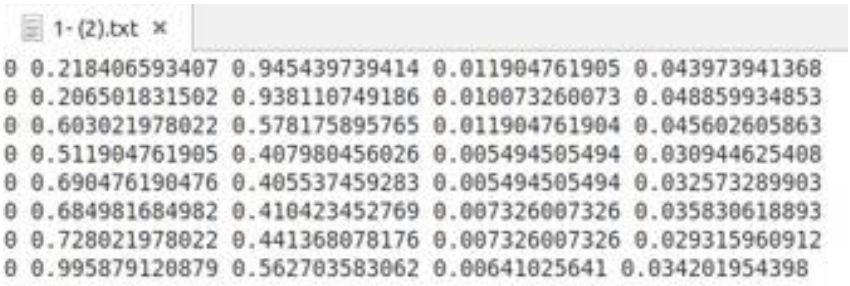

Рисунок 2.3 – Пример текстового документа

Документ содержит 5 показателей: Id, x, y, w, h. Первый из них – Id класса «пешеход» (Id = 0).

Второй и третий показатели – координаты центра ограничивающей рамки вокруг пешехода, полученные по формулам 2.1 и 2.2 соответственно:

$$x = \frac{x_2 + x_1}{2}, \tag{2.1}$$

где x_1 – координата левого верхнего угла ограничивающей рамки по x;

x_2 – координата правого нижнего угла ограничивающей рамки по x.

$$y = \frac{y_2 + y_1}{2}, \tag{2.2}$$

где y_1 – координата левого верхнего угла ограничивающей рамки по y;

y_2 – координата правого нижнего угла ограничивающей рамки по y.

Последние два показателя – ширина и высота рамки, которые вычисляются по формулам 2.3–2.4:

$$w = x_2 - x_1, \tag{2.3}$$

где x_1 – координата левого верхнего угла ограничивающей рамки по x;

x_2 – координата правого нижнего угла ограничивающей рамки по x.

$$h = y_2 - y_1, \tag{2.4}$$

где y_1 – координата левого верхнего угла ограничивающей рамки по y;

y_2 – координата правого нижнего угла ограничивающей рамки по y.

Далее данные были нормализованы путем деления каждого показателя на ширину и высоту всего изображения соответственно (кроме Id класса) (приложение 1). Так как каждый из показателей должен находиться в интервале [0; 1].

2.3 Постановка задачи детекции пешеходов

Основной задачей выпускной квалификационной работы является задача детекции пешеходов по всем направлениям дорожного узла в видеопотоке реального времени. Данная задача очень важна для оценки загруженности пешеходных переходов, предотвращения различных аварий с участием пешеходов.

Пусть X – множество матриц (набор изображений, где каждому $x_i \in X$ соответствуют три матрицы, представляющие собой изображение в цветовом RGB-пространстве), Y – множество значений, где $y_i \in Y$ соответствует набору значений, отвечающих i-му изображению $x_i \in X$ и задающих прямоугольники на нем. Прямоугольник задается набором значений – идентификатор класса, координаты центра прямоугольника, ширина и высота прямоугольника. Целевая функция $f^*: X \to Y$, отображающая множество X на множество Y. Значения целевой функции y^* известны только на конечном множестве пар $(x_i; y_i) -$ тренировочной выборке. Запись $f^*(x_k) = y_k$ означает, что изображение x_k имеет набор значений y_k. Необходимо восстановить функциональную зависимость между изображениями и набором значений, построить алгоритм $B: X \to Y$, обладающий следующими свойствами:

– отображение B должно допускать численную реализацию;

– на тренировочной выборке алгоритм B должен производить правильные значения прямоугольников;

– B должен обладать обобщающей способностью, то есть приближать целевую функцию не только на объектах обучающей выборки, но и на всем множестве X.

Для применения алгоритма B к видеопоследовательности необходимо выполнить ее разбиение на кадры. Аналогичным образом происходит обработка видеопоследовательности, поступающей напрямую с камеры – обработка текущего кадра. К текущему кадру применяется алгоритм B после чего происходит дополнительная обработка – на текущем кадре отрисовываются прямоугольники, полученные с помощью алгоритма B. Таким образом, происходит «подмена» текущего кадра на обработанный с обнаруженными пешеходами.

Стоит отметить, что при решении задачи детекции приходится сталкиваться со следующими факторами, затрудняющими данный процесс:

– часть пешехода может быть невидима (закрыта другими объектами) на изображении;

– условия съемки (освещение, цветовой баланс камеры, искажения изображения, качество изображения) в значительной степени влияют на изображение;

– изменение размера изображения относительно камеры.

Вследствие чего возникает потребность в решении этих проблем. Во многом их можно решить наличием большого набора данных, содержащих пешеходов.

2.4 Метрики качества

2.4.1 Метрика точности ($accuracy$)

Точность ($accuracy$) – это отношение правильно принятых системой решений к общему числу решений (рисунок 2.4). Формально вычисляется по формуле:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \tag{2.5}$$

где TP – количество объектов, верно отнесенных к определенной категории;

FP – количество объектов, ошибочно отнесенных к определенной категории;

FN – количество объектов, ошибочно не отнесенных к определенной категории;

TN – количество объектов, верно не отнесенных к определенной категории.

		Предсказанный	
		true	false
Действительный	true	TP	FN
	false	FP	TN

Рисунок 2.4 – Иллюстрация исходов предсказаний

2.4.2 Метрика точности ($precision$)

Точность ($precision$) системы в пределах класса – это доля ответов, которые действительно принадлежат данному классу относительно всех документов, отнесенных системой к этому классу [20]. Точность рассчитывается по формуле:

$$precision = \frac{TP}{TP + FP}, \tag{2.6}$$

где TP – количество объектов, верно отнесенных к определенной категории;

FP – количество объектов, ошибочно отнесенных к определенной категории.

2.4.3 Метрика полноты

Полнота системы ($recall$) – это доля найденных классификатором ответов, принадлежащих классу относительно всех ответов этого класса в тестовой выборке (рисунок 2.5).

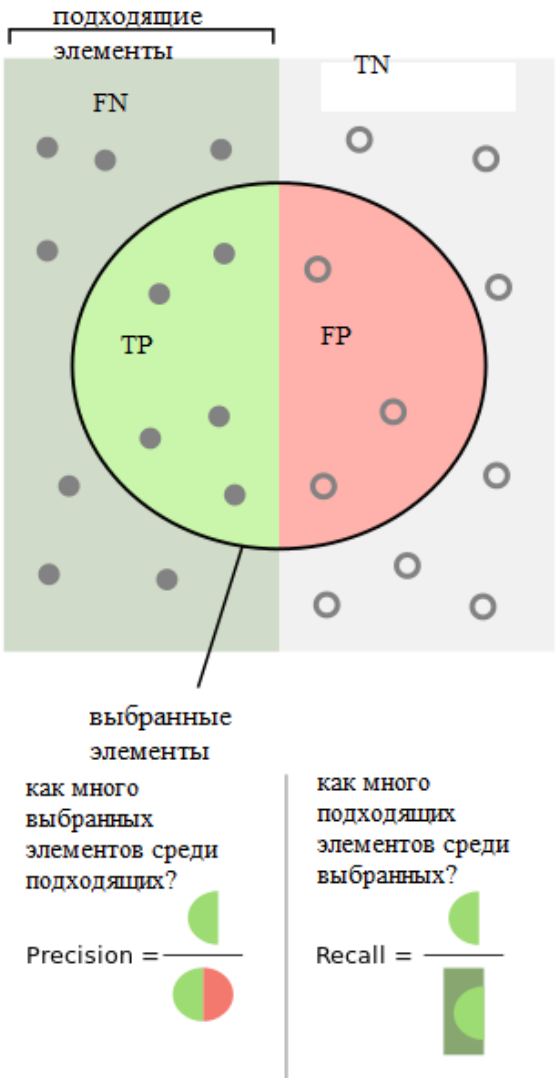

Рисунок 2.5 – Схематическое
представление точности и полноты

Значение полноты вычисляется по формуле:

$$recall = \frac{TP}{TP + FN}, \tag{2.7}$$

где TP – количество объектов, верно отнесенных к определенной категории;

FN – количество объектов, ошибочно не отнесенных к определенной категории.

2.4.4 Метрика локализации объекта

Для оценки правильности заключения объекта в обрамляющее окно используется метрика IoU (рисунок 2.6), которая рассчитывается по формуле:

$$IoU = \frac{|predicted\ box \cap true\ box|}{|predicted\ box\ \cup true\ box|}, \quad (2.8)$$

где $predicted\ box$ – предсказанный ограничивающий прямоугольник для объекта;

$true\ box$ – исходный ограничивающий прямоугольник для объекта.

Типичное пороговое значение, указывающее на правильное обнаружение, составляет $IoU > 50\%$.

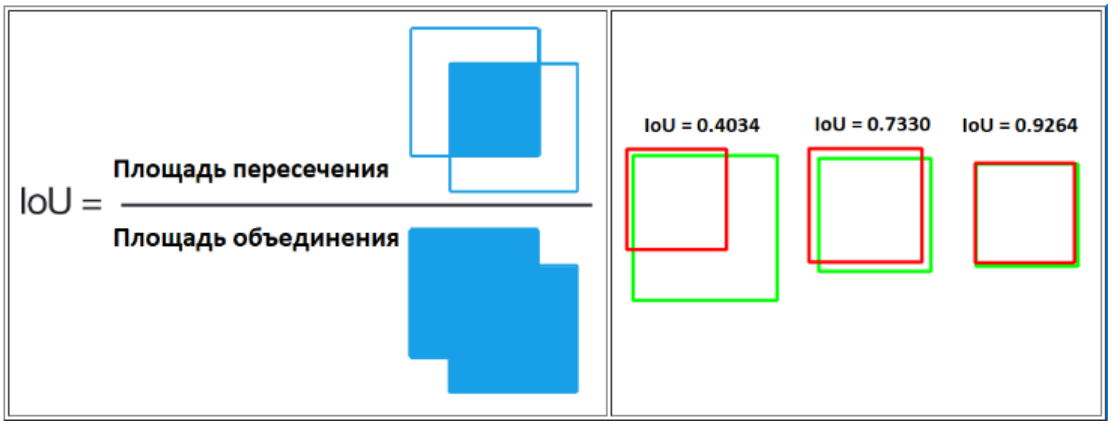

Рисунок 2.6 – Иллюстрация расчета метрики IoU и примеры значений

2.5 Обучение сверточной нейронной сети

2.5.1 Функции активации

Для классификации применяется линейная функция активации:

$$f(x) = x. \quad (2.9)$$

Линейная функция (рисунок 2.7) представляет собой прямую линию и пропорциональна входу (то есть взвешенной сумме на этом нейроне).

Такой выбор активационной функции позволяет получать спектр значений, а не только бинарный ответ.

Однако у этой функции имеется ряд недостатков. Для этой функции производная равна постоянной.

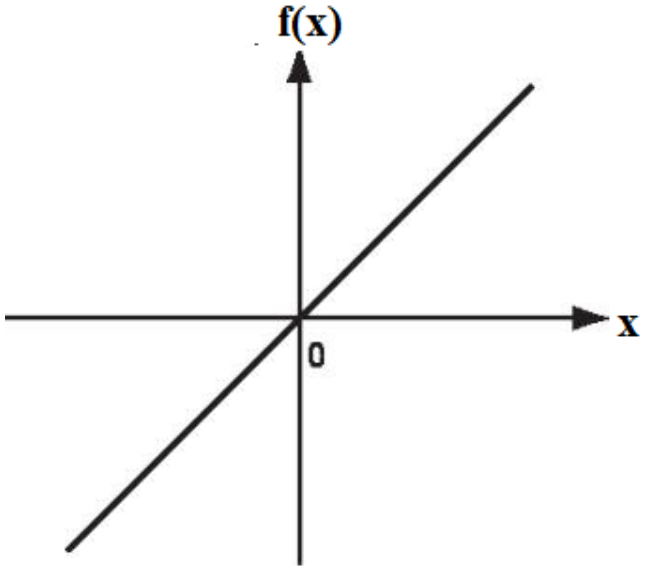

Рисунок 2.7 – Линейная функция активации

Производная от $f(x) = x$ по x равна 1. Это означает, что градиент никак не связан с x. Градиент является постоянным вектором, а спуск производится по постоянному градиенту. Если производится ошибочное предсказание, то изменения, сделанные обратным распространением ошибки, тоже постоянны.

Но существует и другая проблема. Рассмотрим связанные слои. Каждый слой активируется линейной функцией. Значение с этой функции идет в следующий слой в качестве входа, второй слой считает взвешенную сумму на своих входах и, в свою очередь, включает нейроны в зависимости от другой линейной активационной функции [21].

Не имеет значения, сколько слоев мы имеем. Если все они по своей природе линейные, то финальная функция активации в последнем слое будет просто линейной функцией от входов на первом слое.

Для создания нелинейности сети применяется функция активации Leaky ReLU:

$$f(x) = \begin{cases} \alpha x, \text{если } x < 0, \\ x, \text{если } x \geq 0, \end{cases} \quad (2.10)$$

где α – малая константа.

Создание нелинейности сети необходимо, потому что в противном случае было бы не важно количество слоев в сети: их все можно было бы заменить одним единственным слоем с линейной функцией активации.

Так как, функция Leaky ReLU (рисунок 2.8) является модификацией активационной функции ReLU: $(\mathrm{f(x)} = \max(0, \mathrm{x}))$, то она обладает всеми ее преимуществами:

– простота вычисления;

– не подвержена насыщению;

– повышенная скорость сходимости стохастического градиентного спуска.

Но в отличии от ReLU не подвержена выведению из строя при возникновении большого значения градиента.

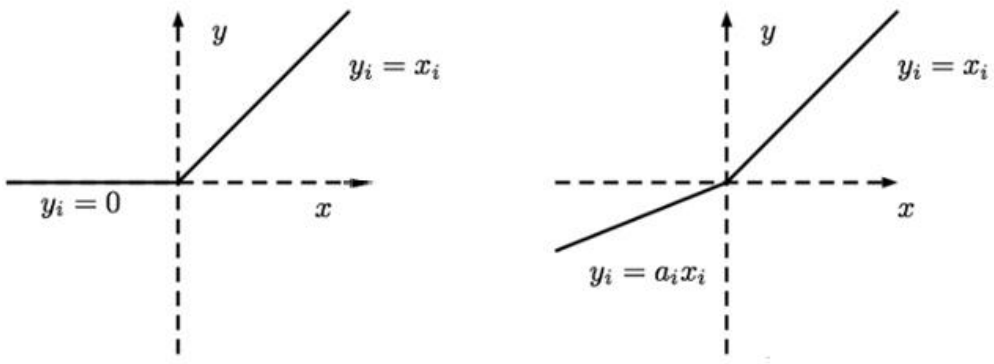

Рисунок 2.8 – Функции активации ReLU и LeakyReLU

2.5.2 Функция потерь

Целью нейронной сети является аппроксимация целевой зависимости $f^*: X \to Y$, где X – множество объектов, Y – множество допустимых решений. Пара (x_i, y_i) называется прецедентом, а совокупность таких пар – обучающей выборкой: $X^n = (x_i, y_i)_{i=1}^n$.

Для оценки качества обучения сети вводится функционал качества:

$$Q(X^n) = \frac{1}{n}\sum_{i=1}^{n} L(a_i, y_i), \tag{2.11}$$

где X^n – тренировочная выборка;

n – число изображений в тренировочной выборке;

y_i – целевой ответ из тренировочного набора данных;

a_i – ответ нейронной сети;

L – функция потерь.

С помощью функции потерь определяется величина ошибки на i-м объекте. Во время обучения нейронная сеть путем подбора весовых коэффициентов минимизирует функционал качества.

В качестве функции потерь при решении данной задачи применяется бинарная кросс-энтропия (формула 2.12). Она позволяет оценить, насколько прогнозируемое распределение a_i близко к истинному распределению y_i.

$$L(a_i, y_i) = -\frac{1}{l}\sum_{j=1}^{l}\left(y_{ij} * \log_2\left(a_{ij}\right) + \left(1 - y_{ij}\right) * \log_2\left(1 - a_{ij}\right)\right), \qquad (2.12)$$

где y_{ij} – j-й прямоугольник истинного распределения;

l – число прямоугольников, соответствующих пешеходам;

a_{ij} – прогнозируемое распределение.

Бинарная кросс-энтропия используется при решении задачи бинарной классификации, когда необходимо определить, является обнаруженный ограничивающий прямоугольник объектом «пешеход» (1) или «не пешеход» (0).

2.5.3 Операция свертки

Свертка – операция над парой матриц A (размера $n_1 \times m_1$) и B (размера $n_2 \times m_2$), результатом которой является матрица $C = A * B$ (рисунок 2.9). Каждый элемент результата вычисляется как скалярное произведение матрицы B и некоторой подматрицы A такого же размера (подматрица определяется положением элемента в результате). То есть определяется по формуле:

$$C_{i,j} = \sum_{u=0}^{n_2-1} \sum_{v=0}^{m_2-1} A_{i+u,j+v} B_{u,v}. \quad (2.13)$$

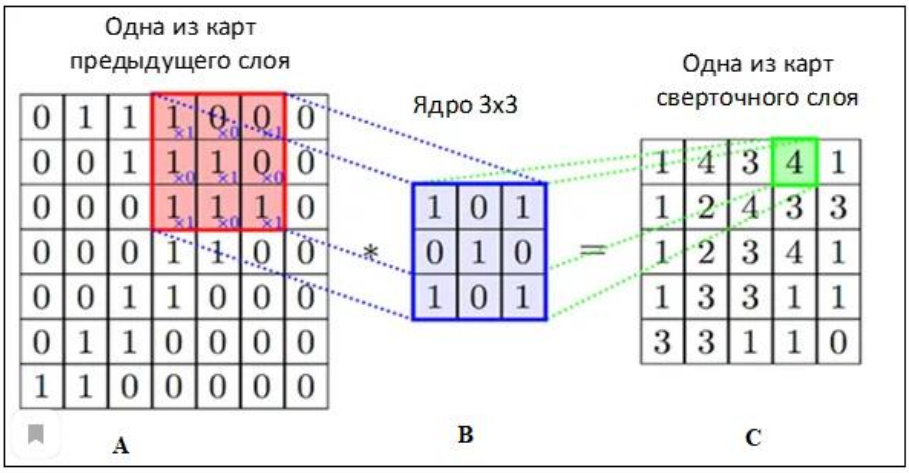

Рисунок 2.9 – Пример операции свертки

Матрицы A и B называют также исходным изображением и ядром свертки (фильтром) соответственно. Основными параметрами свёртки являются размер ядра, сдвиг и дополнение.

На практике, размер ядра, как правило, задается одним числом. Дополнение представляет из себя некоторую «рамку» из нулей вокруг изображения и служит для сохранения размерности матрицы изображения после применения операции дискретной свертки [22]. Сдвиг обычно приравнивают к единице, но также применяется и другой сдвиг – скалярное произведение считается не со всеми возможными положениями ядра, а только с положениями, кратными некоторому сдвигу s.

2.5.4 Операция субдискретизации (пулинга)

В архитектуре CHC обычно применяются слои пулинга между последовательностями свёрточных слоев. Основная задача состоит в последовательном уменьшении пространственных габаритов (разрешения) изображения с намерением уменьшения количества входных параметров для следующего слоя и, соответственно, вычислительных операций в сети, а также контроля обучаемости. Среди видов субдискретизации можно выделить avg-

пулинг, max-пулинг и другие, отличие которых заключается в выборе значения выхода – выбирается среднее значение группы пикселей или выбирается максимальное числовое значение из группы соответственно.

Обычно, каждая карта имеет ядро размером $2x2$, что позволяет уменьшить предыдущие карты сверточного слоя в 2 раза. Вся карта признаков разделяется на ячейки 2x2 элемента, из которых выбираются максимальные по значению. Пример применения операции пулинга представлен на рисунке 2.10.

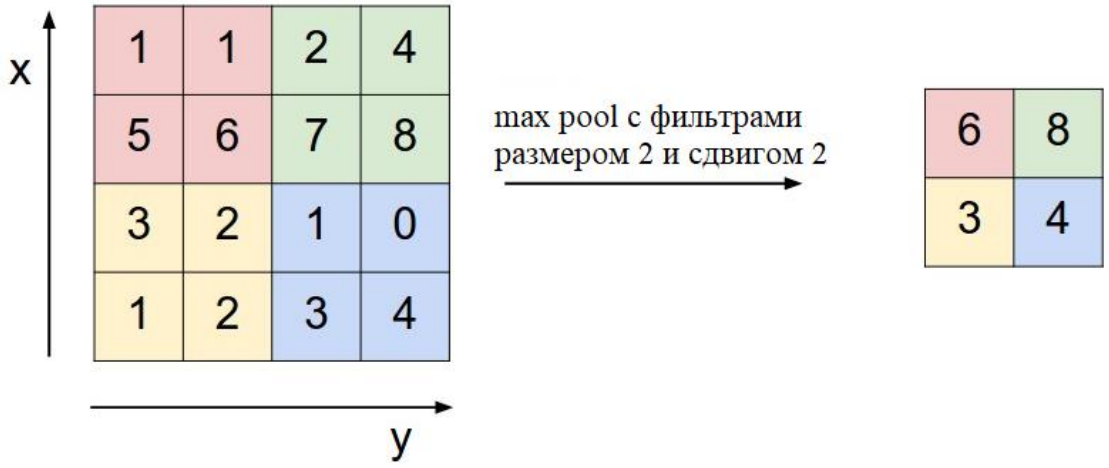

Рисунок 2.10 – Пример операции пулинга с функцией максимума

2.5.5 Кодировщик. Нейронная сеть DarkNet-53. Декодировщик

В качестве кодировщика выбрана нейронная сеть Darknet-53 – модель нейронной сети, предназначенная для решения задачи классификации изображений. Darknet-53 состоит из 53 сверточных слоев с применением функции активации Leaky ReLU и последующей сверткой с шагом 2. Особенностью сети являются специальные остаточные (residual) связи, позволяющие сети быть более глубокой и устойчивой в процессе обучения без снижения качества работы (рисунок 2.11).

На вход кодировщику подается набор изображений (пакет), размерность которых кратна 32 (это обусловлено тем, что кодировщик производит постепенное сжатие изображения до размерности, которая в 32 раза меньше, чем у

исходного изображения). Изображения представляют собой набор матриц, описывающих представление в цветовом пространстве RGB.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8

Рисунок 2.11 – Конфигурация Darknet-53

Основная задача кодировщика – снизить пространственное разрешение изображения постепенным уменьшением его размерности и выделить ряд карт признаков, которые будут представлять изображение на более высоком и глубоком семантическом уровне [23]. Обучение кодировщика происходит с помощью метода обратного распространения ошибки. Модель обучается подбирать весовые коэффициенты карт признаков таким образом, чтобы точность предсказания увеличивалась.

Декодировщик – часть нейронной сети кодировщик-декодировщик, которая восстанавливает изображение исходной размерности из представления низкой размерности, полученной кодировщиком. Как правило, декодировщик идет в паре с кодировщиком, и выходное изображение является преобразованным изображением, поступившим на вход кодировщику.

2.5.6 Метод оптимизации функции потерь

В качестве метода оптимизации функции потерь в работе был использован метод ускоренного градиента Нестерова (NAG). Данный метод является улучшенной версией метода градиентного спуска. Как известно, основным недостатком метода градиентного спуска является проблема застревания в локальных минимумах, эту проблему пытаются решить методы, являющиеся его улучшениями. В методе ускоренного градиента Нестерова вместо того, чтобы оценивать градиент функции потерь в текущей позиции, мы оцениваем градиент сразу в новой позиции. Так мы найдем точку, где окажемся в следующий момент.

Рассмотрим градиентный шаг метода оптимизации NAG.

1. Сначала выполним шаг в направлении истории градиента:

$$w_{temp} = w_t - \rho v_{t-1}, \tag{2.14}$$

где $w_t -$ веса нейронной сети на текущем шаге;

$\rho -$ коэффициент сохранения, соответствует трению, как при движении по твердой поверхности;

$v_{t-1} -$ скорость для перемещения на предыдущем шаге.

2. Далее определим веса нейронной сети на следующем шаге:

$$w_{t+1} = w_{temp} - \eta \nabla L\left(w_{temp}\right), \tag{2.15}$$

где $w_{temp} -$ значение шага в направлении истории градиента;

$\eta -$ параметр скорости обучения;

$L -$ функция потерь.

3. Вычислим значение скорости для перемещения на текущем шаге:

$$v_t = \rho v_{t-1} + \eta \nabla L(w_{temp}), \quad (2.16)$$

где $\rho -$ коэффициент сохранения, соответствует трению, как при движении по твердой поверхности;

$v_{t-1} -$ скорость для перемещения на предыдущем шаге;

$\eta -$ параметр скорости обучения;

$L -$ функция потерь;

$w_{temp} -$ значение шага в направлении истории градиента.

На рисунке 2.12 представлен шаг алгоритма Нестерова.

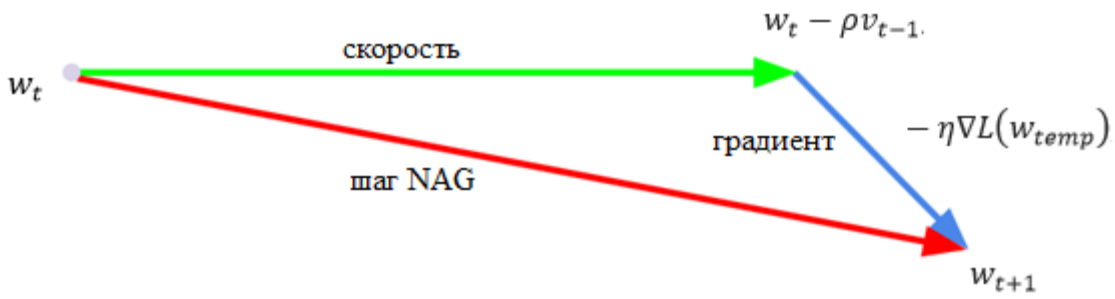

Рисунок 2.12 – Представление работы алгоритма NAG

Стоит заметить, что вместо последовательности из двух шагов (в сторону направления скорости и в сторону отрицательного градиента) алгоритм делает шаг напрямую в следующую точку. Это позволяет нейронной сети сходиться быстрее, поскольку алгоритм делает меньшее число шагов.

Кроме того, объединение идей объединения шагов в сторону изменения скорости и в сторону отрицательного градиента позволяет сети не застревать в седловых точках, а продолжать движение в сторону локального минимума.

2.5.7 Метод обратного распространения ошибки

Обучение нейронной сети будет проводиться по алгоритму обратного распространения ошибки. Опишем метод обратного распространения ошибки на примере многослойного персептрона, так как любая сверточная сеть представима в виде многослойного персептрона и метод идентичен.

Данный алгоритм предполагает два прохода по всем слоям сети: прямой и обратный [24]. При прямом проходе входной вектор подается на входной слой нейронной сети, подвергается модификациям, после чего распространяется по сети от слоя к слою. В результате генерируется набор выходных сигналов, который и является фактической реакцией сети на данный входной образ. Во время прямого прохода все синаптические веса сети фиксированы. Во время обратного прохода синаптические веса настраиваются в соответствии с правилом коррекции ошибок, а именно: фактический выход сети вычитается из желаемого, в результате чего формируется сигнал ошибки. Этот сигнал впоследствии распространяется по сети в направлении, обратном направлению синаптических связей. Отсюда и название – алгоритм обратного распространения ошибки. Синаптические веса настраиваются с целью максимального приближения выходного сигнала сети к желаемому.

Рассмотрим метод обратного распространения ошибки на примере трехслойного персептрона.

На вход сети подаются:

– тренировочный набор $X^n = (x_i; y_i), i \in [1; n], x_i \in \mathbb{R}^n, y_i \in \mathbb{R}^M$, где $n -$ число обучающих пар, $x_i -$ входные данные (изображения), $y_i -$ выходные данные (прямоугольники);

– число нейронов в скрытом слое (H);

– скорость обучения (η).

На выходе получаем веса с входного слоя w_{jh} и веса со скрытого слоя w_{hm}.

Алгоритм будет выглядеть следующим образом.

1. Инициализировать веса и текущую оценку Q.

2. Выбрать $(x_i; y_i) \in X^n,$ где $i \in [1; n]$.

3. Прямой ход:

а) выполним прямой проход через скрытый слой, вычислим значения выходов с нейронов скрытого слоя:

$$u_i^h = \sigma_h\left(\sum_{j=0}^{p} w_{jh}x_i^j\right), \quad (2.17)$$

где $\sigma_h -$ функция активации на h-ом нейроне;

$h -$ номер нейрона в скрытом слое ($h \in [1; H]$);

$w_{jh} -$ веса с входного слоя;

$p -$ число пикселей в изображении;

$x_i^j -$ значение пикселя входного изображения;

б) вычислим производную функции активации σ_h';

в) вычислим предсказания нейронной сети a_i^m:

$$a_i^m = \sigma_m\left(\sum_{h=0}^{H} w_{hm}u_i^h\right), \quad (2.18)$$

где $\sigma_m -$ функция активации на m-ом нейроне;

$m -$ номер нейрона в выходном слое ($h \in [1; M]$);

$w_{hm} -$ веса со скрытого слоя;

$u_i^h -$ значения выходов с нейронов предыдущего скрытого слоя;

г) вычислим производную функции активации σ_m';

д) найдем ошибку сети:

$$\varepsilon_i^m = a_i^m - y_i^m, \quad (2.19)$$

где $a_i^m -$ выход нейросети после прямого прохода;

$y_i^m - m$-й компонент вектора y_i;

е) вычислим функционал качества для i-ой пары тренировочной выборки:

$$Q_i = L(a_i; y_i), \quad (2.20)$$

где $L -$ функция потерь, представленная формулой 2.12;

$a_i -$ ответ нейронной сети;

$y_i -$ правильный ответ для данного изображения x_i.

4. Обратный ход.

Вычислим ошибку сети на h-ом нейроне скрытого слоя:

$$\varepsilon_i^h = \sum_{m=1}^{M} \varepsilon_i^m \sigma'_m w_{hm}, \tag{2.21}$$

где $\varepsilon_i^m -$ ошибка сети m-ом нейроне выходного слоя;

$\sigma'_m -$ производная функции активации на m-ом нейроне;

$w_{hm} -$ веса связей, идущих от скрытого слоя до выходного слоя;

$h -$ номер нейрона в скрытом слое $(h \in [1; H])$.

5. Градиентный шаг:

а) найдем веса на скрытом слое при обратном проходе:

$$w_{hm} = w_{hm} - \eta \varepsilon_i^m \sigma'_m u_i^h, \tag{2.22}$$

где $h -$ номер нейрона в скрытом слое $(h \in [1; H])$;

$m -$ номер нейрона в выходном слое $(h \in [1; M])$;

$\eta -$ скорость обучения;

$\varepsilon_i^m -$ ошибка сети на m-ом нейроне выходного слоя;

$\sigma'_m -$ производная функции активации на m-ом нейроне;

$u_i^h -$ значения выходов с нейронов предыдущего скрытого слоя;

б) найдем веса на входном слое при обратном проходе:

$$w_{jh} = w_{jh} - \eta \varepsilon_i^h \sigma'_h x_i^j, \tag{2.23}$$

где $h -$ номер нейрона в скрытом слое $(h \in [1; H])$;

$j -$ номер нейрона в выходном слое $(j \in [0; n])$;

$\eta -$ скорость обучения;

$\varepsilon_i^h -$ ошибка сети на h-ом нейроне скрытого слоя;

$\sigma'_h -$ производная функции активации на h-ом нейроне;

$x_i^j -$ значение пикселя входного изображения.

6. Оценка функционала:

$$Q(X^n) = \frac{1}{n}\sum_{i=1}^{n} Q_i, \qquad (2.24)$$

где $n -$ число обучающих пар;

$X^n -$ тренировочный набор;

$Q_i -$ функционал качества для i-й пары тренировочной выборки.

Пункты 2–6 алгоритма требуется выполнять до тех пор, пока Q не стабилизируется.

2.6 Выводы по главе 2

В данной главе была описана архитектура сверточной нейронной сети YOLOv3. Были выбраны метрики качества, такие как точность (accuracy), точность (precision), полнота и локализация объекта, которые нужны для дальнейшей оценки работы нейронной сети. Представлено описание сбора данных (изображений), их предварительная обработка для последующего обучения нейронной сети. Была поставлена задача детекции пешеходов. Решение этой задачи необходимо для дальнейшей оценки загруженности пешеходных переходов, ее снижения, предотвращения числа аварий с участием пешеходов. В данной главе был также описан метод обучения нейронной сети YOLOv3 – метод обратного распространения ошибки, его особенности, алгоритм работы. Описаны функции активации – линейная и Leaky Relu, а также – бинарная кросс-энтропия, являющаяся функций потерь.

3 РЕЗУЛЬТАТЫ ДЕТЕКТИРОВАНИЯ, ТРЕКИНГА И ПОДСЧЕТА ПЕШЕХОДОВ В РЕЖИМЕ РЕАЛЬНОГО ВРЕМЕНИ СЕТЬЮ YOLOV3

3.1 Конфигурация нейронной сети

YOLOv3 состоит из магистральной сети DarkNet-53 и содержит 106 слоев, среди которых 75 слоев свертки, остаточные блоки, пропускаемые соединения и слои с повышенной дискретизацией (таблица 3.1). Остаточный блок включает в себя пару сверточных слоев размерностью 3×3 и 1×1 с shortcut соединением, необходимым для предотвращения затухания градиента.

Таблица 3.1 – Конфигурация нейронной сети YOLOv3

Count	Type	Filters	Size/Stride	Input	Output
1	Convolutional	32	$3 \times 3/1$	$992 \times 480 \times 3$	$992 \times 480 \times 32$
	Convolutional	64	$3 \times 3/2$	$992 \times 480 \times 32$	$992 \times 480 \times 64$
	Convolutional	32	$1 \times 1/1$	$992 \times 480 \times 64$	$496 \times 240 \times 32$
	Convolutional	64	$3 \times 3/1$	$496 \times 240 \times 32$	$496 \times 240 \times 64$
	Shortcut				
1	Convolutional	128	$3 \times 3/2$	$496 \times 240 \times 64$	$248 \times 120 \times 128$
2	Convolutional	64	$1 \times 1/1$	$248 \times 120 \times 128$	$248 \times 120 \times 64$
	Convolutional	128	$3 \times 3/1$	$248 \times 120 \times 64$	$248 \times 120 \times 128$
	Shortcut				
1	Convolutional	256	$3 \times 3/2$	$248 \times 120 \times 128$	$124 \times 60 \times 256$
8	Convolutional	128	$1 \times 1/1$	$124 \times 60 \times 256$	$124 \times 60 \times 128$
	Convolutional	256	$3 \times 3/1$	$124 \times 60 \times 128$	$124 \times 60 \times 256$
	Shortcut				
1	Convolutional	512	$3 \times 3/2$	$124 \times 60 \times 256$	$62 \times 30 \times 512$
8	Convolutional	256	$1 \times 1/1$	$62 \times 30 \times 512$	$62 \times 30 \times 256$
	Convolutional	512	$3 \times 3/1$	$62 \times 30 \times 256$	$62 \times 30 \times 512$
	Shortcut				

Продолжение таблицы 3.1

Count	Type	Filters	Size/Stride	Input	Output
1	Convolutional	1024	$3 \times 3/2$	$62 \times 30 \times 512$	$31 \times 15 \times 1024$
4	Convolutional	512	$1 \times 1/1$	$31 \times 15 \times 1024$	$31 \times 15 \times 512$
	Convolutional	1024	$3 \times 3/1$	$31 \times 15 \times 512$	$31 \times 15 \times 1024$
	Shortcut				
3	Convolutional	512	$1 \times 1/1$	$31 \times 15 \times 1024$	$31 \times 15 \times 512$
	Convolutional	1024	$3 \times 3/1$	$31 \times 15 \times 512$	$31 \times 15 \times 1024$
1	Convolutional	68	$1 \times 1/1$	$31 \times 15 \times 1024$	$31 \times 15 \times 68$
	Yolo				
	Route				
	Convolutional	256	$1 \times 1/1$	$31 \times 15 \times 68$	$31 \times 15 \times 256$
	Upsample		$\times 2$	$31 \times 15 \times 256$	$62 \times 30 \times 256$
	Route				
3	Convolutional	256	$1 \times 1/1$	$62 \times 30 \times 256$	$62 \times 30 \times 256$
	Convolutional	512	$3 \times 3/1$	$62 \times 30 \times 256$	$62 \times 30 \times 512$
1	Convolutional	68	$1 \times 1/1$	$62 \times 30 \times 512$	$62 \times 30 \times 68$
	Yolo				
	Route				
	Convolutional	256	$1 \times 1/1$	$62 \times 30 \times 68$	$62 \times 30 \times 128$
	Upsample		$\times 2$	$62 \times 30 \times 128$	$124 \times 60 \times 128$
	Route				
3	Convolutional	128	$1 \times 1/1$	$124 \times 60 \times 128$	$124 \times 60 \times 128$
	Convolutional	256	$3 \times 3/1$	$124 \times 60 \times 128$	$124 \times 60 \times 256$
1	Convolutional	68	$1 \times 1/1$	$124 \times 60 \times 256$	$124 \times 60 \times 68$
	Yolo				

На вход сети подаются изображения размерностью $992px \times 480px$ для дальнейшей их обработки и обучения нейронной сети на них. DarkNet53 отвечает за модификацию этих изображений, их подготовку к следующим этапам и получает на выходе соответствующие карты признаков. После чего эти карты признаков идут на блоки YOLOv3, которые занимаются детекцией.

На слои YOLO поступают уже преобразованные изображения для их дальнейшей обработки. На первом слое YOLO разрешение сетки составляет 1/32 входного изображения, на последнем слое – 1/8 входного изображения, что позволяет обнаруживать как большие, так и мелкие объекты. Между слоями YOLO также находятся слои свертки, route слои, объединяющие массив предыдущих слоев вместе и upsampling слои, предназначенные для увеличения изображения.

3.2 Алгоритмы обучения и тестирования нейронной сети YOLOv3

Для анализа точности полученных результатов работы нейронной сети построен общий алгоритм тестирования нейронной сети YOLOv3, представленный на рисунке 3.1.

Также рассмотрим подробнее вспомогательный алгоритм обучения нейронной сети YOLOv3.

Имеется набор данных X^n (формула 3.1).

$$X^n = \{(x, y) | x \in \mathbb{R}^n, y \in \mathbb{R}^M\}, \quad (3.1)$$

где $(x, y) -$ пара «изображение-прямоугольник».

Изначально происходит разделение входных данных на валидационную (V) и тренировочную (T) выборки в соотношении 20% к 80% соответственно, таким образом тренировочная выборка составляет 67 200 изображений, валидационная – 16 800 изображений.

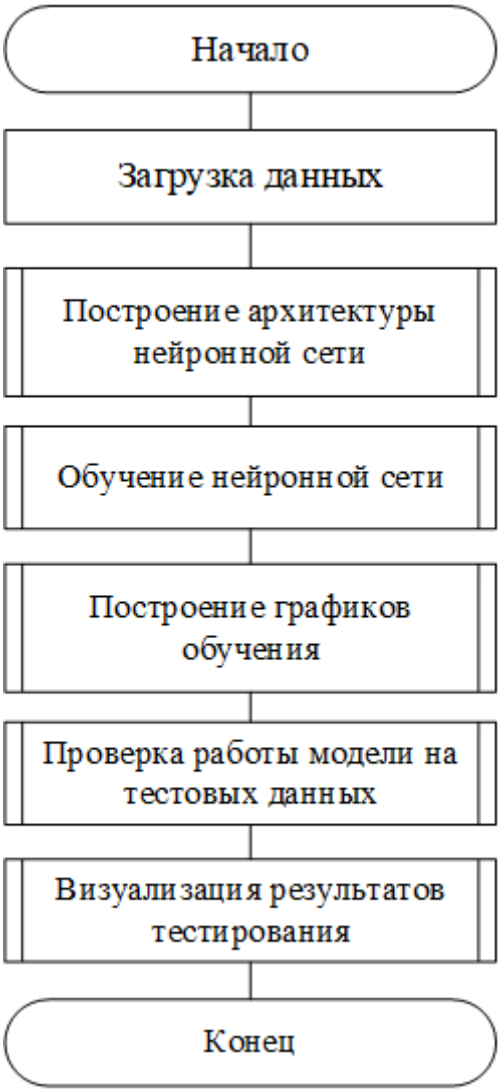

Рисунок 3.1 – Алгоритм
тестирования нейронной сети

После чего начинается цикл обучения на эпохе. За одну эпоху происходит обработка всех изображений [25]. Всего эпох – 15 000. В цикле происходит разделение тренировочной выборки на пакеты (T_1), т.е. число изображений, которые будут обрабатываться нейросетью одновременно, $t_i \in T_1,\ i = \overline{1..N}$, где $N -$ число пакетов. Один пакет содержит 2 изображения. Далее начинается обучение на пакете методом обратного распространения ошибки, который был описан в главе 2.

Изначально происходит прямой проход множества изображений x из пакета t_i через нейронную сеть f^*. Пусть $p = f^*(x)$ – множество предсказаний нейронной сети (прямоугольники). Далее сравниваются набор значений (координаты центра, высота и ширина прямоугольников), полученный после

прямого прохода нейросети с разметкой, составляющей тренировочный набор, и вычисляется ошибка сети (l_1) на пакете. Затем происходит усреднение ошибки на всех изображениях пакета, происходит обновление весов и вычисляется общая ошибка (l_2) на тренировочной выборке, а также вычисляются метрики качества каждые 300 эпох. Алгоритм обучения нейронной сети представлен на рисунке 3.2 (приложение 2).

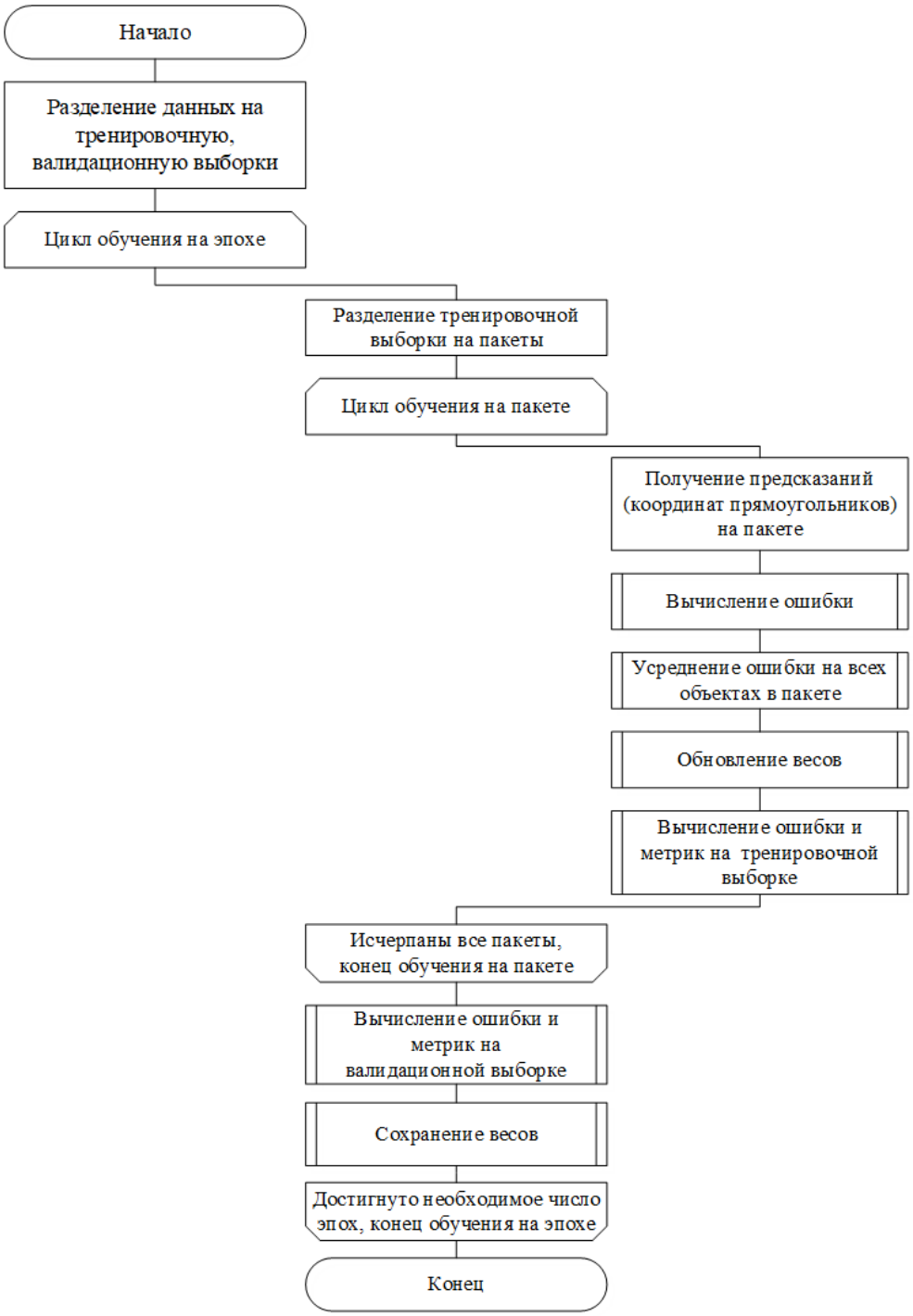

Рисунок 3.2 – Алгоритм обучения нейронной сети

После обработки всех пакетов, происходит вычисление ошибки (l_3) на валидационной выборке, а также каждые 300 эпох рассчитываются метрики качества на этой выборке [26]. Каждые 300 эпох сохраняются полученные веса, чтобы в случае переобучения была возможность использовать не последние сохраненные веса, а промежуточные. После того, как число эпох достигнет 15 000, обучение нейронной сети заканчивается.

3.3 Результаты обучения, полученные метрики

Для оценки качества распознавания объекта «пешеход» были использованы такие метрики как точность ($precision$), точность ($accuracy$), полнота ($recall$) и локализация объекта (IoU) [27].

Во время обучения нейронной сети происходил подсчет метрик качества распознавания на валидационной и обучающей выборках. В результате была получена матрица ошибок, представленная на рисунке 3.1.

Ячейка матрицы, соответствующая числу 153 266, показывает, что 153 266 пешеходов были верно определены нейронной сетью как объект «пешеход». Ячейка матрицы, соответствующая числу 8195, говорит о том, что 8195 пешеходов были классифицированы нейронной сетью, как «не пешеход». Ячейка матрицы, соответствующая числу 7454, показывает, что нейронная сеть классифицировала 7454 объектов, как «пешеход», хотя эти объекты относились к категории «не пешеход». Ячейка матрицы, соответствующая числу 0, говорит о том, что 0 объектов, относящихся к категории «не пешеход», были классифицированы нейронной сетью, как «не пешеход».

Исходя из матрицы ошибок можно представить значения метрик качества распознавания в виде таблицы 3.2.

Таблица 3.2 – Значения метрик для класса «пешеход»

Класс	Точность ($accuracy$)	Точность ($precision$)	Полнота	Локализация объекта
Пешеход	90.7%	95.4%	94.8%	88.7%

Рисунок 3.3 – Матрица ошибок

После обучения был получен график изменения функции потерь (рисунок 3.4), а также графики, соответствующие метрикам качества распознавания (рисунок 3.5).

Рисунок 3.4 – График изменения функции потерь

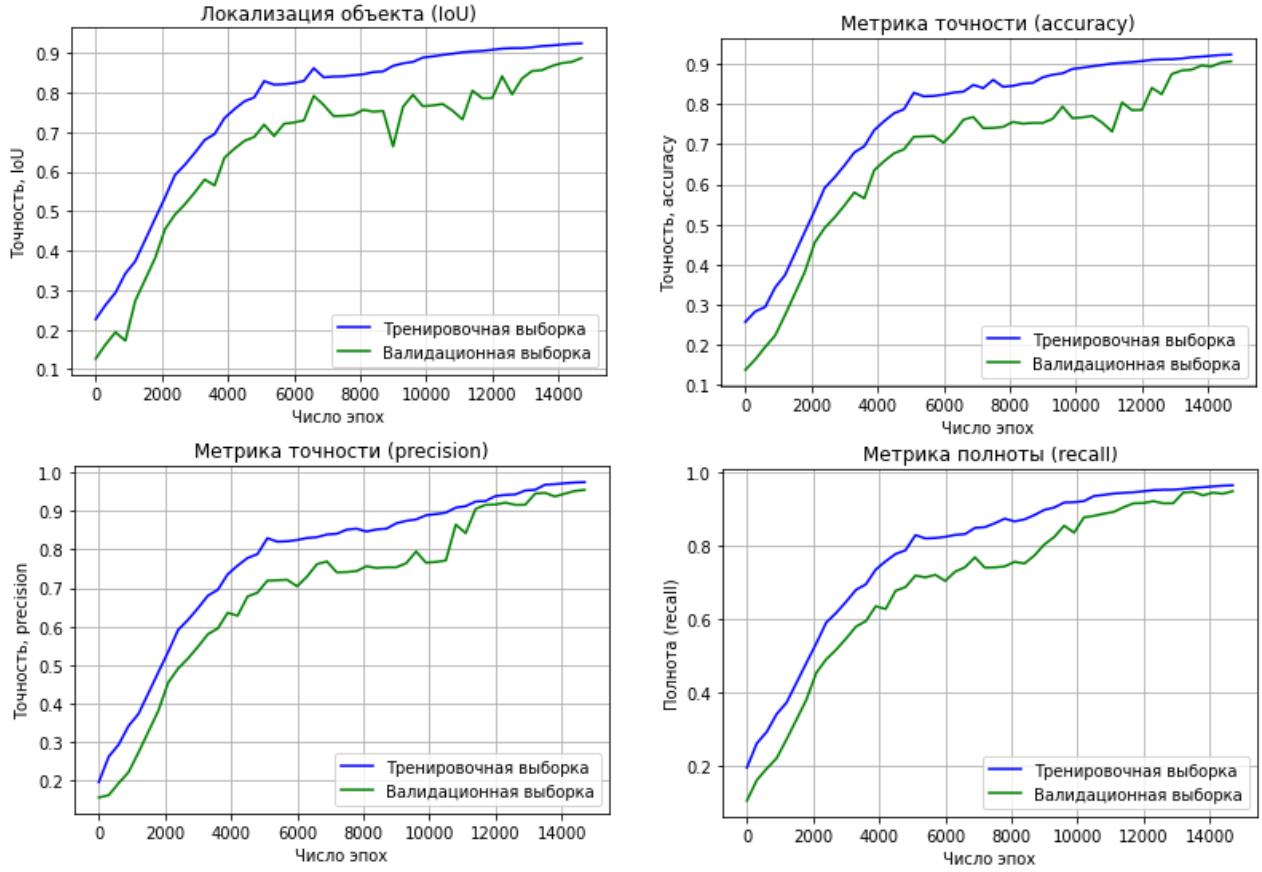

Рисунок 3.5 – Графики метрик качества распознавания

Примеры изображений с распознанными пешеходами представлены на рисунках 3.6–3.7. Код для запуска работы нейронной сети представлен в приложении 3.

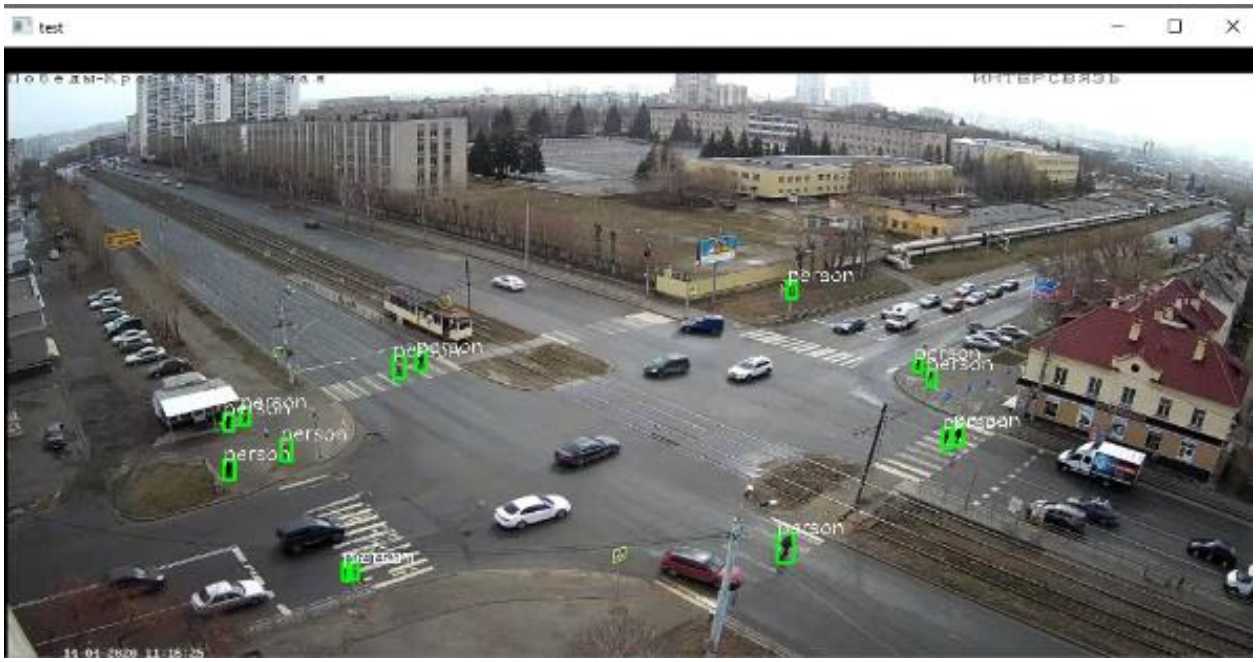

Рисунок 3.6 – Пример результата работы нейронной сети в дневное время

Рисунок 3.7 – Пример результата работы нейронной сети в вечернее время

3.4 Подсчет пешеходов по всем направлениям дорожного узла

Для подсчета пешеходов на перекрестке выделены специальные зоны, в которых ведется подсчет пешеходов [28]. На рисунке 3.8 эти зоны изображены синими четырехугольниками. Таким образом ведется подсчет пешеходов на каждом из пешеходных переходов (приложение 4).

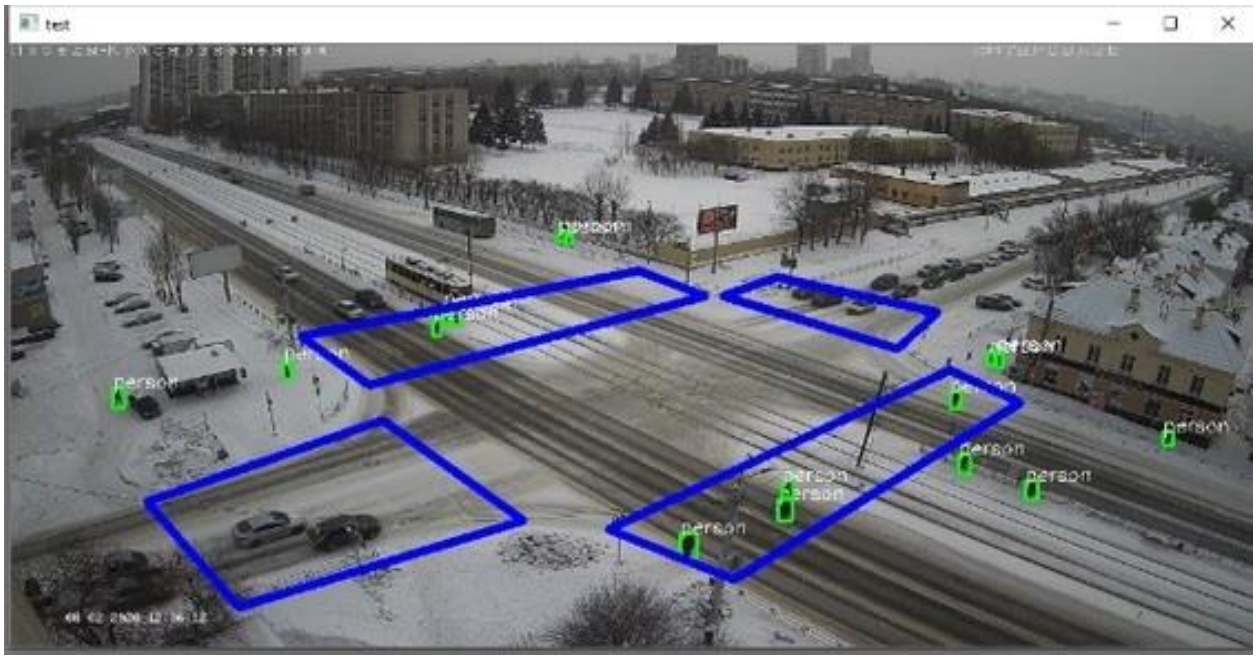

Рисунок 3.8 – Пример результата работы нейронной сети

Опишем процесс подсчета пешеходов. Считается, что, после того, как пешеход пересек четырехугольную зону, число людей, прошедших через данный пешеходный переход, увеличивается [29]. Пусть (x, y) – координаты центра прямоугольника, включающего пешехода. (a, b) – координаты верхнего правого угла четырехугольника. (c, d) – координаты нижнего правого угла четырехугольника. Тогда, если выполняется неравенство 3.2, значит, что пешеход находится по левую сторону от прямой А (рисунок 3.9) и соответствует прямоугольнику 1 на рисунке 3.9.

$$\frac{x-a}{c-a} - \frac{y-b}{d-b} \leq 0. \tag{3.2}$$

Выполнение неравенства 3.3 означает, что пешеход находится по правую сторону от прямой А, 2-й прямоугольник на рисунке 3.7.

$$\frac{x-a}{c-a} - \frac{y-b}{d-b} \geq 0. \tag{3.3}$$

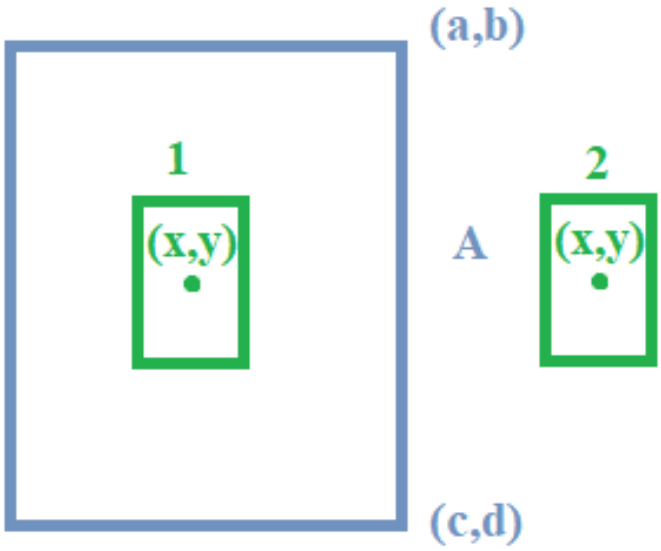

Рисунок 3.9 – Пример положения пешехода
относительно стороны четырехугольника

Таким образом, пешеход находится в определенной четырехугольной зоне, если координаты центра прямоугольника, соответствующего пешеходу, лежат по левую сторону от прямой А и по правую сторону от прямой, противоположной прямой А [30]. Аналогично с двумя другими сторонами

четырехугольной зоны. Значит, после того, как пешеход покинул зону подсчета, мы увеличиваем число человек, прошедших через эту зону.

Для оценки точности подсчета пешеходов был выбран тридцатиминутный видеофрагмент. Оценка точности производилась отдельно по каждому пешеходному переходу [31, 32]. Результаты сравнения исходного числа пешеходов, прошедших через пешеходный переход, с полученным результатом программы представлены на рисунке 3.10.

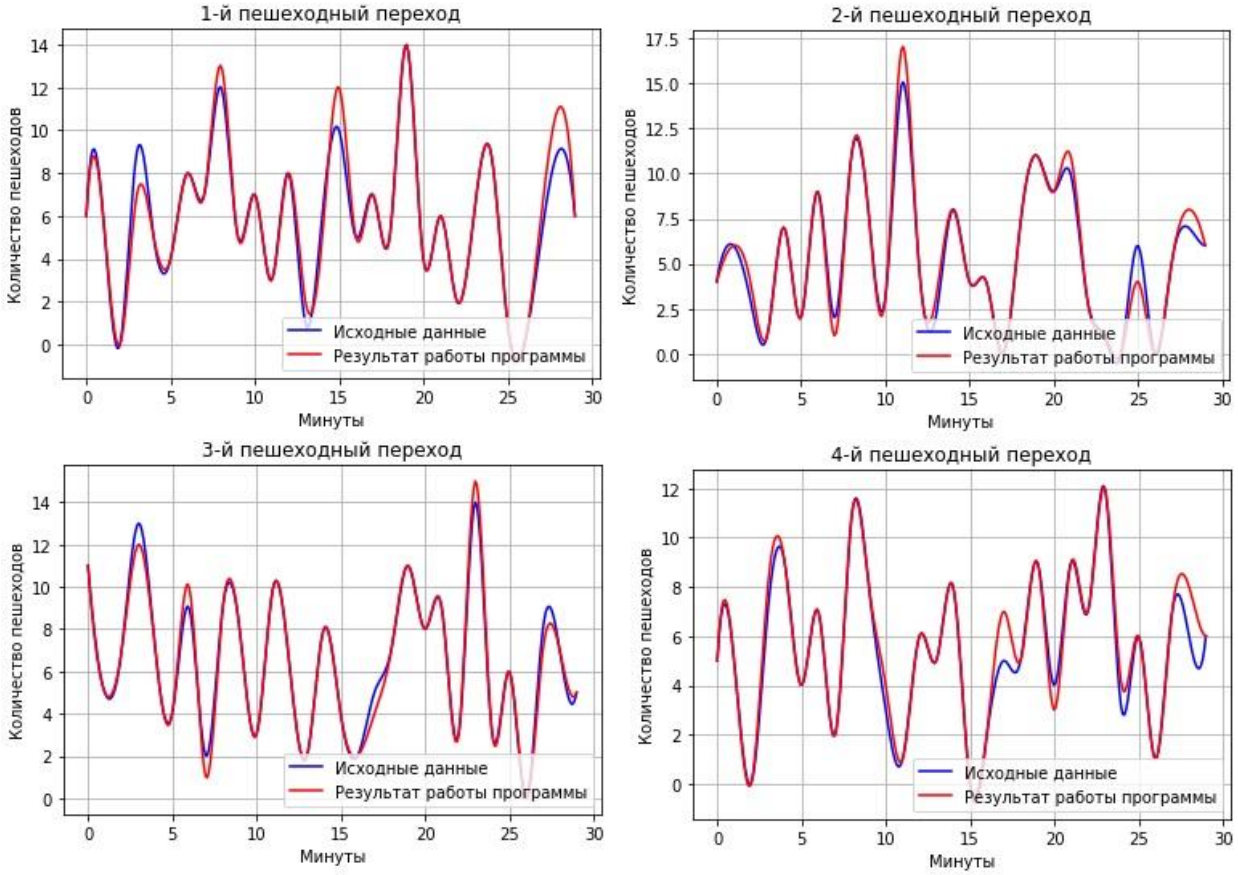

Рисунок 3.10 – Графики точности подсчета пешеходов

Из рисунка видно, что в целом ошибка программы незначительна и составляет не более двух пешеходов в минуту.

3.5 Выводы по главе 3

В данной главе была рассмотрена конфигурация нейронной сети YOLOv3, ее особенности. Также были подробно рассмотрены алгоритмы обучения и тестирования нейронной сети. Была вычислена матрица ошибок,

показывающая насколько полно и точно работает модель. Исходя из которой точность ($accuracy$) равна 90.7%, точность ($precision$) равна 95.4%, полнота ($recall$) равна 94.8% и локализация объекта (IoU) составляет 88.7%.

Были представлены графики обучения YOLOv3, а именно – график функции потерь, отражающий значение ошибки во время обучения нейронной сети, а также графики точности ($accuracy$), точности ($precision$), полноты ($recall$) и локализации объекта (IoU), на которых также отражено, как изменялась точность у различных метрик по мере обучения. Из графиков видно, что точность у всех метрик по мерее обучения возрастала, как на валидационной, так и на тренировочной выборках. График функции потерь убывал, что свидетельствует об уменьшении ошибки во время обучения.

Также были показаны результаты работы нейронной сети в дневное и вечернее время, подробно описан процесс подсчета пешеходов на каждом из пешеходных переходов. Приведены графики точности подсчета для каждой зоны, соответствующей пешеходному переходу, из которых видно, что система подсчета пешеходов работает довольно точно и ошибка программы составляет не более двух пешеходов в минуту.

ЗАКЛЮЧЕНИЕ

Цель данной работы заключалась в разработке модели искусственной нейронной сети, которая в режиме реального времени обнаруживает и ведет подсчет пешеходов в каждом направлении дорожного узла.

В ходе работы задача обнаружения пешеходов была сформулирована, как задача детекции объектов (пешеходов) на изображении, рассмотрены различные подходы к ее решению, которые подразделяются на две основные группы – методы из области компьютерного зрения и глубокие нейронные сети. Была выбрана нейронная сеть YOLOv3 для решения поставленной задачи.

Также была составлена математическая модель и разработана архитектура для решения задачи детекции пешеходов. Были приведены метрики качества и функция потерь, которые позволяют определить точность работы YOLOv3. В качестве алгоритма обучения выбран метод обратного распространения ошибки с применением оптимизации функции потерь методом ускоренного градиента Нестерова.

Для сбора данных для обучения за внимание были взяты 7 камер различных дорожных узлов. С них было собрано около 7000 изображений, на которых размечен каждый из пешеходов. После дальнейших преобразований, путем аугментации, число изображений выросло до 84 000. Среди них 16 800 изображения представляют собой валидационную выборку, оставшиеся – тренировочную. Подготовка данных осуществлялась с помощью платформы Sypervisely.

В качестве фреймворка для обучения использовался DarkNet. Это фреймворк с открытым исходным кодом, написанный на языке Си. В качестве среды разработки использовался PyCharm с применением языка программирования Python 3.6.

Для проведения анализа точности работы нейронной сети были использованы метрики качества, такие как – локализация объекта (IoU), точность

($accuracy$), точность ($precision$), полнота ($recall$), также визуализируется матрица ошибок.

Был подробно описан процесс подсчета пешеходов на каждом из пешеходных переходов. Приведены графики точности подсчета для каждой зоны, соответствующей пешеходному переходу, из которых впоследствии можно увидеть загруженность каждого участка.

Таким образом, цель достигнута, а поставленные задачи – полностью решены. Разработанная архитектура нейронной сети успешно внедрена и используется в проекте «Умный город» (подразделение «Умный транспорт») для обнаружения и подсчета пешеходов по всем направлениям дорожного узла.

В связи со стремительным развитием области нейронных сетей и появлением новых современных решений целью дальнейшей работы может стать исследование новых архитектур и подходов к решению задачи детекции пешеходов.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Волков, С.В. Статистика дорожно-транспортных происшествий в России за текущий год / С.В. Волков // nadoroge.guru – Дата обновления 28.01.2019. URL https://nadoroge.guru/dtp/obschee/statistika-avariy-v-rf(дата обращения: 26.01.2020).

2 Караваев, А.В. Последовательное снижение смертности: МВД опубликовало статистику ДТП за первое полугодие 2019 года/ А.Н. Караваев // russian.rt.com – Дата обновления 15.07.2019. URL https://ru.rt.com/dxt5(дата обращения: 2.02.2020).

3 Олифер, В.Г. Компьютерные сети. Принципы, технологии, протоколы [Текст] / В.Г. Олифер, Н.А. Олифер - СПб: Питер, 2001. – 672 с.

4 Wong, S.-F. Robust Image Segmentation by Texture Sensitive Snake Under Low Contrast Environment/S.-F. Wong, K.-Y.K. Wong // In Proc. Int. Conference on Informatics in Control, Automation and Robotics, 2004. P.430–434.

5 Яшина, М.В. Методы распознавания образов для оценки характеристик пешеходных потоков / М.В. Яшина, А.А. Толмачев // T-Comm: Телекоммуникации и транспорт. – 2017 №8(2). – С. 45–51.

6 Кузнецов, М.К. Методы цифровой обработки видеосигналов // Изв. Южного федерального университета. Технические науки. – 2013 № 11(148). – С. 79–83.

7 Шапиро, Л. Компьютерное зрение / Л. Шапиро. – М.: БИНОМ. Лаборатория знаний, 2013. – 752 с.

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

9 Круглов, В.В. Искусственные нейронные сети. Теория и практика / В.В. Круглов, В.В. Борисов – М.: Горячая линия – Телеком, 2002 – 382 с..

10 Girshick, R. Region-Based Convolutional Networks for Accurate Object Detection and Segmentation [Text] / R. Girshick, J. Donahue, T. Darrell,

J. Malik // IEEE Transactions on Pattern Analysis and Machine Intelligence. – 2015 №28(1). – 16 p.

11 Uijlings, J. Selective Search for Object Recognition [Text] / J.R.R. Uijlings, K.E.A. van de Sande, T. Gevers, A.W.M. Smeulders // International Journal of Computer Vision. – 2013 № 104(2). – P. 154–171.

12 Girshick, R. Fast R-CNN [Text] / R. Girshick // Computer Vision and Pattern Recognition (CVPR'15). - Boston, MA, USA: IEEE, 2015 – 9 p.

13 Ren, S. Faster R-CNN: towards real-time object detection with region proposal networks / He K, Girshick R, et al // International Conference on Neural Information Processing Systems. MIT Press, 2015. P. 91–99.

14 Redmon, J. YOLO: Real-Time Object Detection [Электронный документ] URL:http://pjreddie.com/darknet/yolo/ (дата обращения 7.01.2020).

15 Redmon, J. You Only Look Once: Unified, Real-Time Object Detection [Text] / J. Redmon, S. Divvala, R. Girshick, A. Farhadi // Computer Vision and Pattern Recognition (CVPR'16). - Las Vegas, NV, USA : IEEE, 2016. – 9 p.

16 Redmon, J. YOLOv3: An incremental improvement / J. Redmon, A. Farhadi // Computer Vision and Pattern Recognition (CVPR'18). - Washington, USA : IEEE, 2018. – 6 p.

17 Wei, L. SSD: Single Shot MultiBox Detector [Электронный ресурс] – URL:https://arxiv.org/pdf/1512.02325.pdf (дата обращения 17.01.2020).

18 Lin, T.-Y. Feature Pyramid Networks for Object Detection/ T.-Y. Lin, P. Doll, R. Girshick// Computer Vision and Pattern Recognition (CVPR'14). – Washington, USA: IEEE, 2014. – 9 p.

19 Джонс, М.Т. Программирование искусственного интеллекта в приложениях / М. Т. Джонс. – М.: ДМК Пресс, 2011. – 312 с.

20 Ростовцев, В.С. Искусственные нейронные сети. / В.С. Ростовцев. – Санкт-Петербург: Лань, 2019. – 216 с.

21 Вакуленко, С.А. Практический курс по нейронным сетям: учебное пособие / С. А. Вакуленко, А. А. Жихарева. – Санкт-Петербург: НИУ ИТМО, 2018. – 71 с. – Текст: электронный // Лань: электронно-библиотечная

система. – URL: https://e.lanbook.com/book/136500 (дата обращения: 16.02.2020). – Режим доступа: для авториз. пользователей

22 Созыкин, А.В. Обзор методов обучения глубоких нейронных сетей / А.В. Созыкин // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. – 2017. – № 3. – С. 28–59. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/journal/issue/ 306705 (дата обращения: 09.02.2020). – Режим доступа: для авториз. пользователей.

23 Хайкин, С. Нейронные сети. Полный курс / С. Хайкин. – Москва: Вильямс, 2006. – 1103 с.

24 Vincent, D. A guide to convolution arithmetic for deep learning / D. Vincent, V. Francesco // Computer Vision and Pattern Recognition (CVPR'18). – Italy: IEEE, 2018. – 16 p.

25 Федоров, Д.Ю. Программирование на языке высокого уровня Python: учебное пособие / Д.Ю. Федоров. – 2-е изд. – Москва: Издательство Юрайт, 2020. – 161 с. – Текст: электронный // Юрайт: образовательная платформа. – URL: https://urait.ru/viewer/programmirovanie-na-yazyke-vysoko go-urovnya-python-454101 (дата обращения: 21.03.2020). – Режим доступа: для авториз. пользователей.

26 Маккинни, У. Python и анализ данных/ У. Маккинни; перевод с английского А.А. Слинкина. – 2-е изд., испр. и доп. – Москва: ДМК Пресс, 2020. – 540 с. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/ book/131721 (дата обращения: 05.01.2020). – Режим доступа: для авториз. пользователей.

27 Шелудько, В.М. Основы программирования на языке высокого уровня Python: учебное пособие / В.М. Шелудько. – Ростов-на-Дону, Таганрог: Издательство Южного федерального университета, 2017. – 146 с.

28 Bewley, A. Simple online and realtime tracking [Электронный ресурс] – URL: https://arxiv.org/pdf/1602.00763.pdf (дата обращения 19.04.2020).

29 Dicle, C. The way they move: Tracking multiple targets with similar appearance / C. Dicle, M. Sznaier, O. Camps// International Conference on Computer Vision, 2013. P. 73–84.

30 Олейник, А.Л. Применение бинарных дескрипторов для трекинга множества лиц в системах Научно-технический вестник информационных технологий, механики и оптики / А. Л. Олейник. – М.: ДМК Пресс, 2016. – С. 670–677.

31 Bernardin, K. Evaluating Multiple Object Tracking Performance: The CLEAR MOT Metrics / K. Bernardin, R. Stiefelhagen // International Conference on Computer Vision, 2008. P. 94–107.

32 Rezatofighi, S. Joint Probabilistic Data Association Revisited / S. Rezatofighi, A. Milan, Z. Zhang // International Conference on Computer Vision, 2015. P. 74–87.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ 1

Код нормализации файлов с разметкой

```
#переименование файлов
import io
import os
from os.path import join

path="data/1/"
listImgNames=os.listdir(join(path,"img"))
for i in range(len(listImgNames)):
     os.renames(join(path,"img/"+listImgNames[i]),join("data/1/","14-
("+str(i+1)+").png"))
     os.renames(join(path,"ann/"+listImgNames[i]+".json"),join("data/1
/","14- ("+str(i+1)+").png.json"))

#перевод к нужному формату
import json
import os
import io
from os.path import join

path="data/1/"
listNames=os.listdir(path)
for i in listNames:
     if i.find("json")==-1:
          continue
     with io.open(join(path,i),"r") as file:
          data=json.load(file)
          leftTop=0,0
          rightBot=0,0
          saveFile=io.open(join(path,i.split(".")[0]+".txt"),"w")
          for j in range(len(data['objects'])):

     leftTop=tuple(data['objects'][j]['points']['exterior'][0])

     rightBot=tuple(data['objects'][j]['points']['exterior'][1])
                s="0 "+str(leftTop[0])+" "+str(leftTop[1])+"
"+str(rightBot[0])+" "+str(rightBot[1])+"\n"
                saveFile.write(s.decode('utf-8'))

#удаление лишних файлов
import os
import io
from os.path import join
```

```
path="data/1/"
listNames=os.listdir(path)
for i in listNames:
     if i.find("json")==-1:
          continue
     os.remove(join(path,i))


#нормализация данных1
import json
import os
import io
from os.path import join
from PIL import Image

path="data/1/"
listNames=os.listdir(path)
for i in listNames:
     if i.find("png")==-1:
          continue
     im = Image.open(join(path,i))
     (width, height) = im.size
     coord = []
     with io.open(join(path,i.split(".")[0] + ".txt"),"r") as file:
          for j in file:
               coord.append(j)
     file = io.open(join(path,i.split(".")[0] + ".txt"),"w")
     for j in coord:
          k = j.split(" ")
          s = "0"
          for h in range(len(k)-1):
               if((h + 1) % 2 != 0):
                    s = s + " " + str(float(k[h+1])/float(width))
               else:
                    s = s + " " + str(float(k[h+1])/float(height))
          file.write(s.decode('utf-8') + u"\n")
     file.close()

#нормализация данных2
import json
import os
import io
from os.path import join
from PIL import Image

path="data/1/"
listNames=os.listdir(path)
for i in listNames:
     if i.find("png")==-1:
          continue
     im = Image.open(join(path,i))
```

```
      (width, height) = im.size
      coord = []
      with io.open(join(path,i.split(".")[0] + ".txt"),"r") as file:
            for j in file:
                  coord.append(j)
      file = io.open(join(path,i.split(".")[0] + ".txt"),"w")
      for j in range(len(coord)):
            k = coord[j].split(" ")
            s = "0"
            for h in range(len(k)-1):
                  if((h + 1) <= 2):
                        s = s + " " + str(k[h + 1])
                  elif((h + 1) > 2):
                        s = s + " " + str(float(k[h + 1]) - float(k[h -
1]))
            file.write(s.decode('utf-8') + u"\n")
      file.close()


#нормализация данных3
import json
import os
import io
from os.path import join
from PIL import Image

path="data/12/"
listNames=os.listdir(path)
for i in listNames:
      if i.find("png")==-1:
            continue
      im = Image.open(join(path,i))
      (width, height) = im.size
      coord = []
      with io.open(join(path,i.split(".")[0] + ".txt"),"r") as file:
            for j in file:
                  coord.append(j)
      file = io.open(join(path,i.split(".")[0] + ".txt"),"w")
      for j in range(len(coord)):
            k = coord[j].split(" ")
            s = "0"
            for h in range(len(k)-1):
                  if((h + 1) <= 2):
                        s = s + " " + str(float(k[h + 1]) + float(k[h +
3])/2)
                  elif((h + 1) > 2):
                        s = s + " " + str(k[h + 1])
            file.write(s.decode('utf-8'))
      file.close()
```

```
#разделение данных на тренировочную и валидационную выборки
import io
import PIL
from PIL import Image,ImageDraw
import os
from os.path import join

path="data/ALL_YOLODATA/"
listNames=os.listdir(path)
c=1
def addname(namePict,a):
    if a%5==0:
        file=io.open("data/valid","a+")
        pathToImg = join('data/ALL_YOLODATA/', namePict + '\n')
        file.write(pathToImg)
        file.close()
    else:
        file=io.open("data/fileslist","a+")
        pathToImg = join('data/ALL_YOLODATA/', namePict + '\n')
        file.write(pathToImg)
        file.close()
for i in listNames:
     if i.find("txt")!=-1:
           continue
     addname(i.decode('utf-8'),c)
     c=c+1
```

ПРИЛОЖЕНИЕ 2

Обучение нейронной сети

```
#команда для запуска обучения
./darknet detector train data/train.data cfg/yolov3.cfg darknet53.conv.74


#конфиг файл с архитектурой нейронной сети
[net]
# Testing
#batch=1
#subdivisions=1
# Training
batch=64
subdivisions=32
width=992
height=480
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1
letter_box=1

learning_rate=0.001
burn_in=1000
max_batches = 20000
policy=steps
steps=16000,18000
scales=.1,.1

[convolutional]
batch_normalize=1
filters=32
size=3
stride=1
pad=1
activation=leaky

# Downsample

[convolutional]
batch_normalize=1
filters=64
size=3
stride=2
pad=1
```

```
activation=leaky

[convolutional]
batch_normalize=1
filters=32
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=64
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

# Downsample

[convolutional]
batch_normalize=1
filters=128
size=3
stride=2
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=64
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear
```

```
[convolutional]
batch_normalize=1
filters=64
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

# Downsample

[convolutional]
batch_normalize=1
filters=256
size=3
stride=2
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear
```

```
[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
```

```
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear


[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
```

```
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

# Downsample

[convolutional]
batch_normalize=1
filters=512
size=3
stride=2
pad=1
activation=leaky

[convolutional]
batch_normalize=1
```

```
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear


[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear


[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
```

```
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear


[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear
```

```
[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear


[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
```

```
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

# Downsample

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=2
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1024
```

```
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

######################
```

```
[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=1024
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=1024
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=1024
activation=leaky

[convolutional]
size=1
stride=1
```

```
pad=1
filters=18
activation=linear


[yolo]
mask = 6,7,8
anchors = 4,7  3,6,  2,5  3,5  2,4
classes=1
num=5
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1


[route]
layers = -4

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[upsample]
stride=2

[route]
layers = -1, 61



[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=512
activation=leaky
```

```
[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=512
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=512
activation=leaky

[convolutional]
size=1
stride=1
pad=1
filters=18
activation=linear


[yolo]
mask = 3,4,5
anchors = 4,7  3,6,  2,5  3,5  2,4
classes=1
num=5
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1
```

```
[route]
layers = -4

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[upsample]
stride=2

[route]
layers = -1, 36



[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=256
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=256
activation=leaky
```

```
[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=256
activation=leaky

[convolutional]
size=1
stride=1
pad=1
filters=18
activation=linear


[yolo]
mask = 0,1,2
anchors = 4,7  3,6,  2,5  3,5  2,4
classes=1
num=5
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=0
```

ПРИЛОЖЕНИЕ 3

Код для запуска работы нейронной сети

```
#запуск работы нейронной сети
import operator
from ctypes import *
import math
import random
import os
from queue import Queue
from PIL import Image,ImageDraw
from os.path import join

import cv2
import numpy as np
import time
import darknet
from accidents import AccidentCounter
from database import UpdateVisionData, fill_dict, fill_speeddict,
TaskThreadQueue, InsertAccidentsCount, \
    UpdateAccidentsCount, UpdateSpeedData
import settings

from sort import *

from shapely.geometry import Point
from shapely.geometry.polygon import Polygon

class_to_color = {
    1: (26, 66, 180),
}
def draw_rect(image, space_x, space_y, c):
    cv2.rectangle(image, (space_x[0], space_y[0]), (space_x[1],
space_y[1]), class_to_color[c], 1)
    return image

def convertBack(x, y, w, h):
    xmin = int(round(x - (w / 2)))
    xmax = int(round(x + (w / 2)))
    ymin = int(round(y - (h / 2)))
    ymax = int(round(y + (h / 2)))
    return xmin, ymin, xmax, ymax

def parse_yolo_data(path, ext):
    files = os.listdir(path)
    ext_files = [i for i in files if ext in i]
    if len(ext_files) != 1:
        raise Exception("File *%s must be a single" % ext)
    return path + ext_files[0]
```

```
YOLO_DATA_PATH = "yolo_data/people/"

def YOLO():
    global metaMain, netMain, altNames, YOLO_DATA_PATH, mask
    configPath = parse_yolo_data(YOLO_DATA_PATH, '.cfg')
    metaPath = parse_yolo_data(YOLO_DATA_PATH, '.data')
    weightPath = parse_yolo_data(YOLO_DATA_PATH, '.weights')

    if not os.path.exists(configPath):
        raise ValueError("Invalid config path `" +
                         os.path.abspath(configPath) + "`")
    if not os.path.exists(weightPath):
        raise ValueError("Invalid weight path `" +
                         os.path.abspath(weightPath) + "`")
    if not os.path.exists(metaPath):
        raise ValueError("Invalid data file path `" +
                         os.path.abspath(metaPath) + "`")
    if netMain is None:
        netMain = darknet.load_net_custom(configPath.encode(
            "ascii"), weightPath.encode("ascii"), 0, 1)  # batch size
= 1
    if metaMain is None:
        metaMain = darknet.load_meta(metaPath.encode("ascii"))
    if altNames is None:
        try:
            with open(metaPath) as metaFH:
                metaContents = metaFH.read()
                import re
                match = re.search("names *= *(.*)$", metaContents,
                                  re.IGNORECASE | re.MULTILINE)
                if match:
                    result = match.group(1)
                else:
                    result = None
                try:
                    if os.path.exists(result):
                        with open(result) as namesFH:
                            namesList =
namesFH.read().strip().split("\n")
                            altNames = [x.strip() for x in namesList]
                except TypeError:
                    pass
        except Exception:
            pass
url = "rtsp://tdstream.is74.ru/p/pobkras"
cap = cv2.VideoCapture(url)
darknet_image = darknet.make_image(darknet.network_width(netMain),

darknet.network_height(netMain), 3)
mot_tracker = Sort()
```

```
while True:
            try:
                ret, frame_read = cap.read()
                if frame_read is None or not ret:
                    cap = cv2.VideoCapture(url)
                    continue
                frame_rgb = cv2.cvtColor(frame_read,
cv2.COLOR_BGR2RGB)
                frame_resized = cv2.resize(frame_rgb,

(darknet.network_width(netMain),

darknet.network_height(netMain)),

interpolation=cv2.INTER_LINEAR)
                frame_src = frame_resized.copy()
            except:
                cap = cv2.VideoCapture(url)
                continue
darknet_frame = frame_resized

darknet.copy_image_from_bytes(darknet_image, darknet_frame.tobytes())
            detections = darknet.detect_image(netMain, metaMain,
darknet_image, thresh=0.1)
            det = []
            dets = np.array(det)
            for detection in detections:
                x, y, w, h, score = detection[2][0], \
                                    detection[2][1], \
                                    detection[2][2], \
                                    detection[2][3], \
                                    detection[1]
                xmin, ymin, xmax, ymax = convertBack(
                    float(x), float(y), float(w), float(h))
                pt1 = (xmin, ymin)
                pt2 = (xmax, ymax)
                det.append([xmin, ymin, xmax, ymax, score])

                dets = np.array(det)
          trackers = mot_tracker.update(dets)
          for x1, y1, x2, y2, obj_id, dx1, dy1, dx2, dy2 in trackers:
                __x1, __y1, __x2, __y2 = x1, y1, x2, y2
          draw_rect(frame_resized, [int(__x1), int(__x2)], [int(__y1),
          int(__y2)], det_full[indexx[0], 3])
          image = frame_resized
          image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
          cv2.imshow('frame', image)
          ch = 0xFF & cv2.waitKey(1)
                  if ch == 27:
                      break
     cap.release()
```

```
if __name__ == "__main__":
    try:
        YOLO()
    except Exception as e:
        print(str(e))
        exit(1)
```

ПРИЛОЖЕНИЕ 4

Код для подсчета пешеходов

```
#подсчет пешеходов
import cv2
import math
import numpy as np
from sort import *
count_pedestrian=[0,0,0,0]
current_pedestrian=[[],[],[],[]]
lb = [(408, 379), (181, 448), (108, 364), (296, 299), (408, 379)]
lt = [(553, 200), (285, 271), (231, 232), (500, 180), (553, 200)]
rb = [(804, 285), (576, 426), (476, 386), (748, 259), (804, 285)]
rt = [(738, 213), (706, 242), (567, 200), (610, 185), (738, 213)]

def convertBack(x, y, w, h):
    xmin = int(round(x - (w / 2)))
    xmax = int(round(x + (w / 2)))
    ymin = int(round(y - (h / 2)))
    ymax = int(round(y + (h / 2)))
    return xmin, ymin, xmax, ymax

def is_left(point,point1,point2):
    res=((point[0]-point1[0])/(point2[0]-point1[0])-(point[1]-
point1[1])/(point2[1]-point1[1]))
    return res
def in_rectangle(p):
    global lb, lt, rb, rt
    center=(((p[2]-p[0])/2)+p[0],((p[3]-p[1])/2)+p[1])
    if is_left(center,lb[0],lb[1])>=0 and
is_left(center,lb[2],lb[3])>=0 and is_left(center,lb[1],lb[2])<=0 and
is_left(center,lb[3],lb[4])<=0:
        return 1
    if is_left(center,lt[0],lt[1])>=0 and
is_left(center,lt[2],lt[3])>=0 and is_left(center,lt[1],lt[2])<=0 and
is_left(center,lt[3],lt[4])<=0:
        return 2
    if is_left(center,rt[0],rt[1])>=0 and
is_left(center,rt[2],rt[3])>=0 and is_left(center,rt[1],rt[2])<=0 and
is_left(center,rt[3],rt[4])<=0:
        return 3
    if is_left(center,rb[0],rb[1])>=0 and
is_left(center,rb[2],rb[3])>=0 and is_left(center,rb[1],rb[2])<=0 and
is_left(center,rb[3],rb[4])<=0:
        return 4
    return 0


def count(trackers,img):
```

```
    global lb,lt,rb,rt,count_pedestrian,current_pedestrian
    c_p=[[],[],[],[]]
    for t in trackers:
        lt1 = (int(t[0]),int(t[1]))
        rb1 = (int(t[2]),int(t[3]))
        img = cv2.rectangle(img,lt1,rb1,(0,255,0),2)

cv2.putText(img,"Person",lt1,cv2.FONT_HERSHEY_SIMPLEX,0.5,[0,0,255],1)


    img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
    for i in range(4):
        img=cv2.line(img,lb[i],lb[i+1],(255,0,0),5)
        img=cv2.line(img,lt[i],lt[i+1],(255,0,0),5)
        img=cv2.line(img,rb[i],rb[i+1],(255,0,0),5)
        img=cv2.line(img,rt[i],rt[i+1],(255,0,0),5)

    for t in trackers:
        p=in_rectangle(t)
        if p==0:
            continue
        if p==1:
            c_p[0].append(t[4])
        if p==2:
            c_p[1].append(t[4])
        if p==3:
            c_p[2].append(t[4])
        if p==4:
            c_p[3].append(t[4])
    for i in range(4):
        for j in c_p[i]:
            if j not in current_pedestrian[i]:
                count_pedestrian[i]+=1
        current_pedestrian[i].extend(c_p[i])

    cv2.putText(img, 'lb:%s'%count_pedestrian[0], (30,30),
cv2.FONT_HERSHEY_SIMPLEX, 1, [0, 0, 255], 1)
    cv2.putText(img,'lt:%s'%count_pedestrian[1], (130,30),
cv2.FONT_HERSHEY_SIMPLEX, 1, [0, 0, 255], 1)
    cv2.putText(img, 'rt:%s'%count_pedestrian[2], (230,30),
cv2.FONT_HERSHEY_SIMPLEX, 1, [0, 0, 255], 1)
    cv2.putText(img, 'rb:%s'%count_pedestrian[3], (330,30),
cv2.FONT_HERSHEY_SIMPLEX, 1, [0, 0, 255], 1)

    cv2.imshow("test",img)
```