МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра Прикладная математика и информатика
(наименование)

02.03.03 Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки, специальности)

Технология программирования
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему Реализация алгоритма идентификации объектов на изображении для уточнения информации о возгораниях

Студент Я.А. Рыбаков
(И.О. Фамилия) (личная подпись)

Руководитель к.ф.-м.н., С.В. Баумгертнер
(ученая степень, звание, И.О. Фамилия)

Тольятти 2020

Аннотация

Тема бакалаврской работы: «Реализация алгоритма идентификации объектов на изображении для уточнения информации о возгораниях».

Данная работа выполнялась по заказу в рамках работы центра IT-Students.

В данной бакалаврской работе исследуется вопрос применения сверточных нейронных сетей для обнаружения объектов на изображениях с возгораниями для уточнения информации об этих возгораниях.

Объект исследования - процесс обнаружения объектов на изображении свёрточной нейронной сетью.

Предмет исследования - алгоритм идентификации объектов на изображении для уточнения информации о возгораниях.

Цель исследования – реализовать алгоритм идентификации объектов на изображении для уточнения информации о возгораниях с применением свёрточных нейронных сетей.

Структура работы состоит из введения, трёх разделов, заключения и списка источников.

Во введении дается характеристика предметной области, определяется актуальность исследования, формулируются цель и задачи.

В первом разделе производится анализ применения технологии свёрточных нейронных сетей, их теоретические основы и особенности.

Во втором разделе рассматривается архитектура выбранной для реализации модели свёрточной нейронной сети.

В третьем разделе предоставляется реализация модели для обнаружения объектов, её оценка и тестирование.

В заключении подводятся итоги проделанной работы.

В работе использовано 3 таблицы, 36 рисунков, список литература состоит из 25 литературных источников. Общий объём выпускной квалификационной работы составляет 82 страницы.

Abstract

The present graduation work is devoted to implementing an algorithm for detecting objects in an image to clarify information about fire.

The key issue of the graduation work is the use of convolutional neural networks to detect objects in images showing fire to clarify information about it.

The object of the graduation work is the process of detecting objects in the image by means of the convolutional neural network.

The subject of the graduation work is the algorithm for detecting objects in the image to clarify information about fire.

The goal of the investigation is to implement the algorithm for detecting objects in the image to clarify the information about fires using convolutional neural networks.

The graduation work may be divided into several logically connected parts which are introduction, 3 chapters, conclusions and a list of references.

The introduction describes the subject area, establishes the relevance of the research, as well as sets the goal and objectives.

The first chapter analyzes the application of the convolutional neural network technology, its theoretical foundations and features.

The second chapter dwells on the architecture of the convolutional neural network model to be implemented.

The third chapter covers implementation of the model designed for detecting objects and provides its evaluation and testing.

In conclusion, we would like to underline that the implemented neural network can be used in Unmanned Aerial Vehicle patrol systems to detect fire.

Содержание

Введение

Обнаружение объектов является ключевой способностью систем компьютерного зрения классифицировать и локализовать различные объекты на изображениях и видео. Методы, которые используют системы компьютерного зрения для выполнения задачи обнаружения объектов, делятся на две обширные категории: традиционные методы и эволюционные методы [1]. Традиционные методы обнаружения объектов осуществляют обнаружение нескольких объектов на одном изображении, рассматривая такие признаки, как форма, текстура, цвет. С появлением таких подходов, как свёрточные нейронные сети (СНС) и глубокое обучение, множество объектов могут обнаруживаться на большом количестве изображений и видео. Методы, основанные на СНС и глубоком обучении, известны как эволюционные методы. Ключевая способность эволюционных методов заключается в том, что они могут быть использованы для обнаружения объектов в реальном времени и работать с большими наборами данных, содержащими миллионы изображений [2].

Целью бакалаврской работы является реализация алгоритма идентификации объектов на изображении для уточнения информации о возгораниях. На протяжении работы необходимо будет реализовать модель свёрточной нейронной сети способную распознавать и локализовывать объекты на изображениях, на которых запечатлены возгорания и пожары различного характера, а именно следующие объекты: люди, машины, автобусы и мотоциклы. Данные объекты обнаруживаются с целью уточнения информации о произошедшем возгорании для составления полной картины произошедшей чрезвычайной ситуации.

Для достижения поставленной цели необходимо выполнить следующие задачи:

– проанализировать применение свёрточных нейронных сетей для распознавания и локализации объектов на изображениях;

– разобрать базовую архитектуру свёрточных нейронных сетей;

– подобрать наиболее подходящую по точности и производительности модель свёрточной нейронной сети;

– реализовать выбранную модель;

– провести оценку реализованной модели;

– провести тестирование модели на различных изображениях.

Объектом исследования бакалаврской работы является процесс обнаружения объектов на изображении свёрточной нейронной сетью, предметом исследования – алгоритм идентификации объектов на изображении для уточнения информации о возгораниях.

Бакалаврская работа состоит из введения, трёх разделов, заключения и списка источников.

В первом разделе рассматриваются основы методов машинного обучения, производится анализ технологии искусственных нейронных сетей, после чего рассматриваются математические методы, структур и особенности свёрточных нейронных сетей. Во втором разделе описывается архитектура выбранной модели свёрточной нейронной сети. Разбираются её преимущества для решения поставленных задач. Рассматриваются применяемые методы, обеспечивающие точность и производительность. В третьем разделе объясняется выбор языка программирования и фреймворка для реализации модели. Описываются основные аспекты реализации и обучения модели. Проводится оценка точности работы модели и её тестирование.

Реализованная нейронная сеть может быть использована в системах беспилотного патрулирования местности с помощью летательных дронов для выявления чрезвычайных ситуаций. Получая изображения, на которых было распознано возгорание, сеть будет обнаруживать перечисленные ранее объекты, что позволит получить полную информацию о возгорании, которая поможет принять решение о необходимых средствах для скорейшей ликвидации чрезвычайного происшествия.

1 Анализ применение свёрточных нейронных сетей для локализации и распознавания объектов на изображениях

1.1 Анализ алгоритмов машинного обучения

Основу программы будет составлять искусственная нейронная сеть, которая является одним из алгоритмов машинного обучения, поэтому необходимо определить теоретические основы разделов, на которых базируется технология нейронных сетей. Для этого необходимо начать с основного раздела - машинное обучение.

Машинное обучение - это наука о том, как заставить компьютеры обучаться и действовать так, как это делают люди, предоставляя им знания посредством данных, наблюдений и взаимодействия с миром, а также дать им возможность совершенствовать свое обучение с течением времени автономно [1]. Вышеуказанное определение включает в себя идеальную цель или конечную цель машинного обучения, выраженную многими исследователями в данной области. Фундаментальной целью алгоритмов машинного обучения является обобщение, выходящее за рамки учебных примеров, то есть успешная интерпретация данных, которые они никогда раньше «не видели». Алгоритм обучения - это вычислительная процедура, которая по обучающей выборке производит настройку модели [1]. Выходом алгоритма обучения является функция, аппроксимирующая неизвестную (восстанавливаемую) зависимость. В задачах классификации аппроксимирующую функцию принято называть классификатором (classifier), в задачах восстановления регрессии - функцией регрессии [23]. Существует много различных типов алгоритмов машинного обучения и каждый день публикуются новые алгоритмы, обычно они классифицируются либо по стилю обучения, либо на основе сходства формы или функции.

Рассмотрим классификацию по стилю обучения. Существуют различные способы, которыми алгоритм может моделировать проблему, основываясь на его взаимодействии с опытом, окружающей средой, или на

том, что называется входными данными. Есть несколько основных стилей или моделей обучения, которые может иметь алгоритм. Рассмотрим их с несколькими примерами алгоритмов и типов проблем, к решению которых они подходят. Существует три различных стиля обучения: обучение с учителем, обучение без учителя, обучение с частичным привлечением учителя [23].

При обучении с учителем входные данные называются обучающими данными и имеют известные метки классов или результат, например, спам или не спам для писем, или курс акций в определенное время [1]. Модель подготавливается в процессе обучения, в котором требуется делать предсказания, и корректируется, когда эти предсказания оказываются ошибочными. Процесс обучения продолжается до тех пор, пока модель не достигнет желаемого уровня точности на данных для обучения. Примерами проблем для решения которых подходят алгоритмы данного типа являются классификация и регрессия. Примеры алгоритмов: логистическая регрессия и нейронная сеть обратного распространения.

При обучении без учителя входные данные не размечены и не имеют известного результата [1]. Модель подготавливается путем нахождения взаимосвязей во входных данных. Это может быть извлечение общих правил или математический процесс, направленный на систематическое сокращение избыточности, или группировка данных по схожести. Примерами проблем, которые решают такие алгоритмы являются кластеризация, снижение размерности и поиск ассоциативных правил. Примеры алгоритмов: алгоритм Apriori и k-means (метод k-средних).

При обучении с частичным привлечением учителя входные данные представляют собой комбинацию размеченных и не размеченных примеров [1]. Модель должна как изучать взаимосвязи во входных данных для их организации, так и делать предсказания, после которых корректироваться. Примерами решаемых проблем являются классификация и регрессия. Примерами алгоритмов являются расширения к другим гибким методам,

которые делают предположения о том, как моделировать неразмеченные данные.

Алгоритмы часто группируются по схожести их функций (принцип их работы). Например, методы, основанные на деревьях, или методы, вдохновленные нейронными сетями [1]. Далее рассмотрим алгоритмы, сгруппированные по данному принципу, которые будут использованы для разработки программы.

Искусственные нейронные сети - это модели, вдохновленные структурой и функцией биологических нейронных сетей [1]. Они обычно используется при решении задач регрессии и классификации, но на самом деле это большой класс, состоящим из сотен алгоритмов и вариаций для решения всех типов проблем. Наиболее популярные алгоритмы искусственных нейронных сетей: персептрон, многослойный персептрон, обратное распространение, стохастический градиентный спуск, нейронная сеть Хопфилда.

Методы глубокого обучения - это современное обновление искусственных нейронных сетей [22]. С помощью них строятся гораздо более крупные и сложные нейронные сети. Многие из этих методов работают с очень большими наборами аналоговых размеченных данных, таких как изображения, текст, аудио и видео. Самые популярные алгоритмы глубокого обучения: свёрточные нейронный сети, рекуррентные нейронные сети, сети долгой краткосрочной памяти, глубокие сети доверия и другие.

В данном подразделе были проанализированы алгоритмы машинного обучения. Так как основная задача программы распознавать и локализовать объекты на изображениях, то в качестве алгоритма машинного обучения для реализации была выбрана свёрточная нейронная сеть, которая специализируется на работе с большими наборами аналоговых данных, в том числе и изображений. Далее подробнее остановимся на анализе технологии искусственных нейронных сетей.

1.2 Анализ технологии искусственных нейронных сетей

Искусственная нейронная сеть (ИНС) - это вычислительная система, состоящая из ряда простых, тесно связанных между собой вычислительных элементов, которые в совокупности обрабатывают информацию, поступающую на вход системе [18]. ИНС - это математическая модель и её программная или аппаратная реализация, которые моделируются на основе нейронной структуры коры головного мозга млекопитающих, но в гораздо меньших масштабах [1]. Большая ИНС может состоять из сотен и тысяч вычислительных единиц, в то время как мозг млекопитающего имеет миллиарды нейронов, величина взаимодействия которых соответственно увеличивается. Хотя исследователей ИНС, как правило, не волнует вопрос о том, точно ли их сети похожи на биологические системы, некоторые из них беспокоятся об этом. Например, исследователи точно смоделировали функцию сетчатки и достаточно хорошо смоделировали глаз [1]. Также некоторые ученные пытаются создать цифровой мозг полностью идентичный живому организму, начиная, естественно, с малых представителей царства животных, таких как черви.

Искусственные нейронные сети обладают следующими характеристиками: нелинейность, отображение входной информации в выходную (input-output mapping), очевидность ответа (evidential response), адаптивность, контекстная информация, отказоустойчивость (fault tolerance), масштабируемость, единообразие анализа и проектирования, аналогия с нейробиологией [1].

Искусственный нейрон может быть, как линейным, так и нелинейным. Если нейронная сеть построена из нелинейных нейронов, то она является нелинейной [1]. При чем эта нелинейность распределена по всей сети, так как каждый нейрон, её составляющий нелинейный. Это свойство очень важно, особенно когда физический механизм, генерирующий входной сигнал, является нелинейным, например, речь человека.

ИНС обладают возможностью адаптироваться к различным изменениям окружающей среды. Так, сети, обученные в одной среде, могут быть переобучены для работы в среде с небольшими колебаниями параметров [1]. Существует возможность переобучения уже обученных ИНС, путем оставления базовых слоев и переобучения крайних слоев на новых данных [18]. Также могут быть созданы сети, способные адаптироваться в реальном времени, но в этом вопросе главное соблюсти баланс, так как адаптивная нейронная система с быстро изменяющимися параметрами также может быстро реагировать и на посторонние шумы, что плохо скажется на производительности [18].

ИНС являются довольно отказоустойчивыми благодаря их распределенной структуре [1]. Если один из нейронов будет давать сбой или его связь будет нарушена это не внесет критической ошибки в работу нейронной сети. Снижение качества работы нейронной сети происходит медленно, по мере выхода из строя её элементов. Исходя из этого можно сказать, что только серьёзные повреждения структуры сети кардинально повлияют на её работу.

Нейронные сети, как правило, организованы слоями. Слои состоят из ряда взаимосвязанных узлов, которые содержат функцию активации [2]. Нейронная сеть состоит из 3-х основных частей (видов слоев): входной слой, скрытые слои и выходной слой. Схема простейшей искусственной нейронной сети представлена на рисунке 1. Входной слой – это буквально слой, который вводит информацию для обработки нейронной сетью. Каждая узел этого слоя представляет собой 1 признак (часть информации) [2]. Это может быть что угодно. Это может быть площадь в квадратных метрах вашего дома для программы прогнозирования цен на дом, или значение пикселя на экране для программы, реализующей компьютерное зрение.

Скрытые слои - эти слои выполняют всю обработку для нейронных сетей. Вы можете создать их в своей сети столько, сколько захотите [18]. Как правило, чем больше скрытых слоев у вас есть, тем более точной будет

нейронная сеть. Каждый слой состоит из узлов, имитирующих нейроны мозга живого организма. Эти узлы получают информацию от узлов предыдущего слоя, умножают ее на вес, а затем добавляют к ней смещение. Каждая линия на диаграмме (рисунок 1) представляет собой связь, у которой есть собственный вес – некоторое числовое значение.

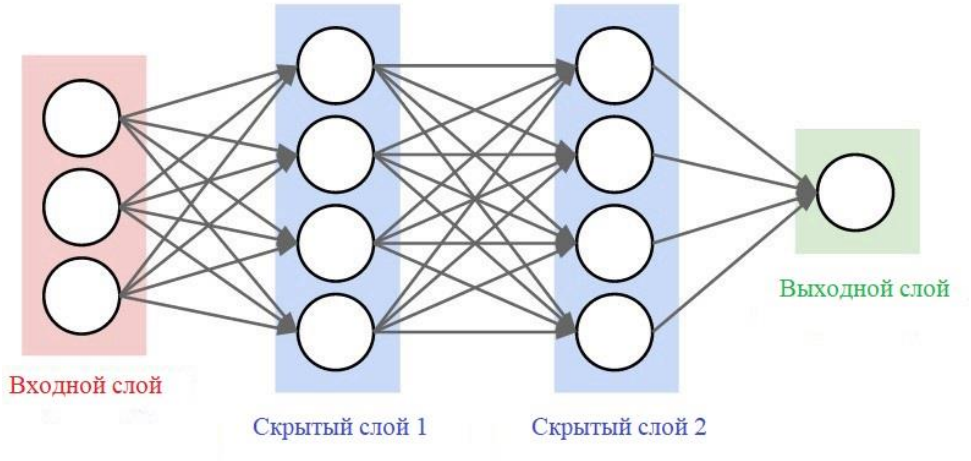

Рисунок 1 – Схема простейшей ИНС

Выходной слой - этот слой просто собирает воедино информацию с последнего скрытого слоя нейронной сети для вывода итоговой информации [2]. Подводя итог, можно сказать, что нейронные сети берут информацию с входного слоя, обрабатывают ее в скрытых слоях и выводят необходимую информацию в выходной слой. Весь этот процесс называется прямым распространением.

Рассмотрим процесс прямого распространения в сети подробнее на примере нелинейного искусственного нейрона, модель которого отображена на рисунке 2. Основные элементы модели: набор связей, сумматор, функция активации и смещение.

Каждая связь или синапс в наборе характеризуется свои весом, проходя по данной связи вход нейрона умножается на вес связи. Синаптический вес связи искусственного нейрона может быть, как отрицательным значением, так и положительным, в отличии от синапсов мозга живого организма [18].

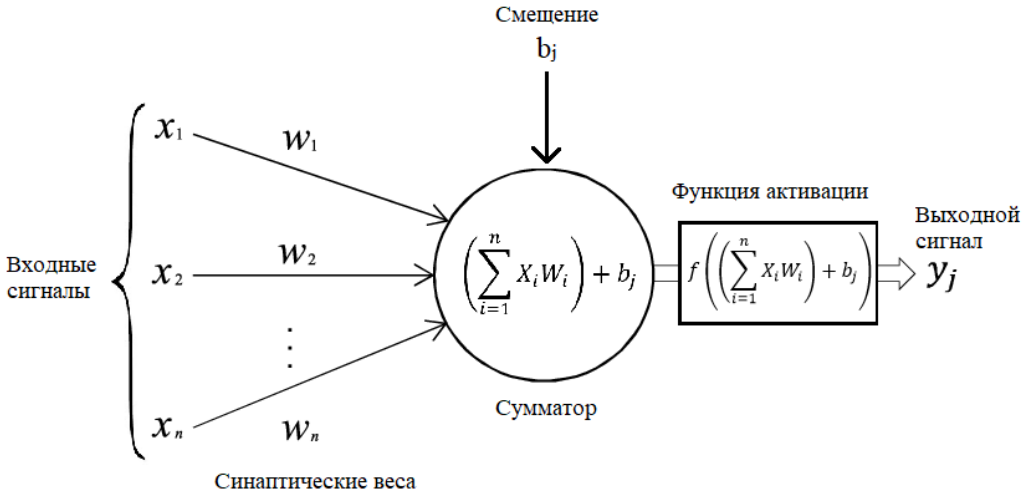

Рисунок 2 – Нелинейная модель искусственного нейрона

Сумматор выполняет сложение входных сигналов, которые предварительно были взвешены относительно соответствующих связей нейрона. По сути, эту операцию можно назвать линейной комбинацией [2].

Смещение представляет собой дополнительный нейрон, входящий в состав каждого слоя предварительной обработки и хранящий значение «1» [2]. Узлы смещения не связаны ни с одним из предыдущих слоев, но обрабатываются так же, как и остальные. Смещение является фундаментальным аспектом большинства машинных методов обучения по нескольким основным причинам:

– без узла смещения ни один слой не смог бы предоставить выходной сигнал для следующего слоя, отличающийся от 0, если бы значения входных сигналов были бы равны 0;

– узлы смещения помогают сетям решать больше проблем разных типов, позволяя им использовать более сложные логические операции;

– смещение служит как пороговое значение в методах глубокого обучения.

Функции активации нейронных сетей являются важнейшим компонентом глубокого обучения [1]. Функции активации определяют выход модели глубокого обучения, ее точность, а также вычислительную эффективность обучения модели. Функции активации - это математические

уравнения, определяющие выход нейронной сети. Функция привязывается к каждому нейрону в сети и определяет, должен ли он быть активирован или нет, исходя из того, имеет ли значение, данный вход нейрона для предсказания. Функции активации также помогают нормировать выход каждого нейрона в диапазон от 0 до 1 или от -1 до 1. Дополнительным аспектом функций активации является то, что они должны быть вычислительно эффективными, так как они вычисляются для тысяч или даже миллионов нейронов для каждого примера данных [18].

Функция активации - это математические "врата" между входным сигналом, поступающим в нейрон, и его выходом, отправляющимся к следующему слою [2]. Она может быть простой, как ступенчатая функция, которая включает или выключает выход нейрона, в зависимости от правила или порога. Или же это может быть преобразование, которое отображает входные сигналы в выходные, необходимые для функционирования нейронной сети. Чаще всего в нейронных сетях используются нелинейные функции активации, которые могут помочь им изучать сложные данные и предоставлять точные прогнозы.

На рисунке 3 представлены графики различных функций активации, которые могут быть использованы при построении модели нейронной сети. Функции активации в основном можно разделить на три типа: линейные, нелинейные и пороговые. График линейной функции представлены на рисунке 3 под пунктом «г». Как видно, функция представляется из себя прямую линию. Поэтому выходные значения функции не будут ограничены каким-либо диапазоном. Это никак не поможет со сложными или различными параметрами данных, которые подаются в нейронные сети.

График функции двоичного шага, которая является пороговой функцией представлен под пунктом «а» рисунка 3. Если входное значение выше или ниже определенного порога, нейрон активируется и посылает точно такой же сигнал следующему слою. Проблема с такими функциями активации состоит в том, что они не позволяют использовать многозначные

выходы - например, они не могут производить классификацию входных сигналов в одну из нескольких категорий. График знаковой функции активации представлен на рисунке 3 под пунктом «е», она также является пороговой функцией.

Нелинейные функции активации в основном разделяются на основании их диапазона или кривых [18]. Логистическая сигмоидальная функция активации является нелинейной, её график отображен на рисунке 3 под пунктом «б». Значения данной функции вычисляются по формуле (1).

$$f(x) = \frac{1}{1 + e^{-x}} \quad (1)$$

где $f(x)$ – это функция активации;

e – основание натурального логарифма, математическая константа;

x – переменная, для которой находится значения функции активации.

Основная причина, по которой используется сигмоидальная функция, заключается в том, что она выдает значения в диапазоне от 0 до 1. Поэтому она особенно часто используется в моделях, где необходимо предсказать вероятность некоторого выходного сигнала. Так как вероятность какого-либо события существует только между 0 и 1, сигмоидальная функция будет правильным выбором. Функция является дифференцируемой, монотонной, но её производная нет. Использование сигмоидальной функции может привести к тому, что нейронная сеть застрянет во время тренировки [3].

График функции активации гиперболический тангенс отображен на рисунке 3 под пунктом «д». Эта функция похожа на логистическую сигмоидальную функцию, но лучше. Диапазон значений функции от -1 до 1. Преимущество заключается в том, что отрицательные входные значения будут отображены в виде отрицательных, а нулевые будут отображены в виде близких к нулю на графике. Функция дифференцируемая, монотонная, её производная не монотонная. В основном функция используется для классификации между двумя классами [18].

В настоящее время ReLU (rectified linear unit, усеченное линейное преобразование) является наиболее используемой функцией активации в мире, так как она используется почти во всех свёрточных нейронных сетях или при глубоком обучении [2]. График этой функции представлен на рисунке 3 под пунктом «в». Функция и её производная являются монотонными.

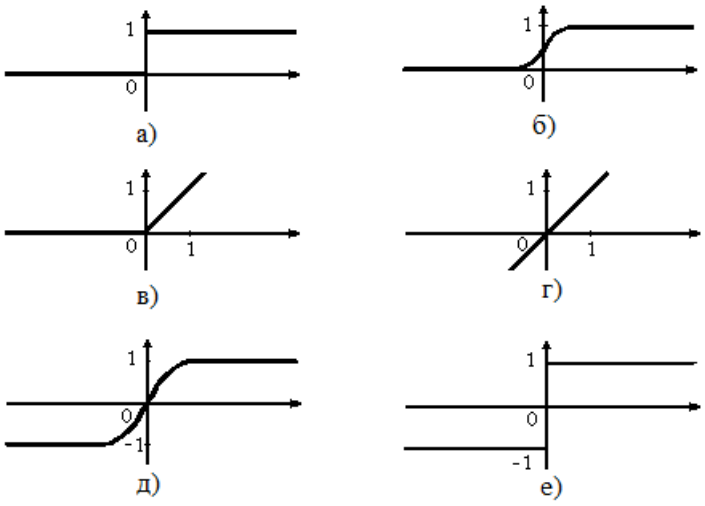

Рисунок 3 – Графики функций активации нейрона

Рассмотрим процесс обучения искусственной нейронной сети. Обучение нейронной сети, то есть нахождение оптимальных значений параметров (весов связей и смещений) является частью глубокого обучения [4]. Можно рассматривать этот процесс обучения как итеративный: проходящий вперед и обратно по слоям нейронной сети. Прохождение вперед называется прямым распространением информации, а обратно – обратным распространением информации.

Первая фаза прямого распространения состоит в следующем: в нейронную сеть поступают обучающие данные, которые проходят через всю нейронную сеть и на выходе сеть выдает рассчитанные ей предсказания (метки) [3]. То есть происходит передача входных данных по сети таким образом, чтобы все нейроны сети применили свое преобразование к информации, которую они получили от нейронов предыдущего слоя, и

отправили ее нейронам следующего уровня. Когда данные пройдут через все слои, и все нейроны произведут свои вычисления, последний слой будет достигнут, и результатом его вычислений будут предсказания меток для подступивших в сеть входных примеров.

Далее мы используем функцию потерь для оценки потерь (ошибки) нейронной сети, а также для сравнения и измерения того, насколько хорошим или плохим был результат предсказания сети по отношению к правильному результату, учитывая то, что используется обучение с учителем, и у каждого примера есть метка, которая содержит ожидаемое значение [4]. В идеале необходимо стремиться к тому, чтобы ошибка была как можно ближе к нулю, то есть с минимальным расхождением между рассчитанным и ожидаемым значением. Поэтому, по мере обучения модели, веса связей между нейронами будут постепенно корректироваться до тех пор, пока не будут получены оптимальные результаты предсказаний [3].

Функция потерь используется для оптимизации значений параметров в модели нейронной сети. Функция потерь отображает набор значений параметров сети в скалярное значение, которое показывает, насколько хорошо этот параметр соответствует задаче, которую должна выполнять сеть [4]. Проблема оптимизации заключается в минимизации функции потерь.

После вычисления потерь необходимо распространить эту информацию в обратном направлении. Поэтому этот этап называется обратным распространением ошибки. Начиная с выходного слоя, эта информация о потерях распространяется на все нейроны скрытого слоя, которые вносили непосредственный вклад в выходной сигнал [3]. Однако при распространении нейроны скрытого слоя получают лишь часть общего сигнала потерь, исходя из относительного вклада каждого нейрона в выходной сигнал. Этот процесс повторяется, от слоя к слою, до тех пор, пока все нейроны в сети не получат сигнал о потерях, описывающий их относительный вклад в суммарную ошибку. Объединив все вышесказанное, получим схему этапов, представленную на рисунке 4.

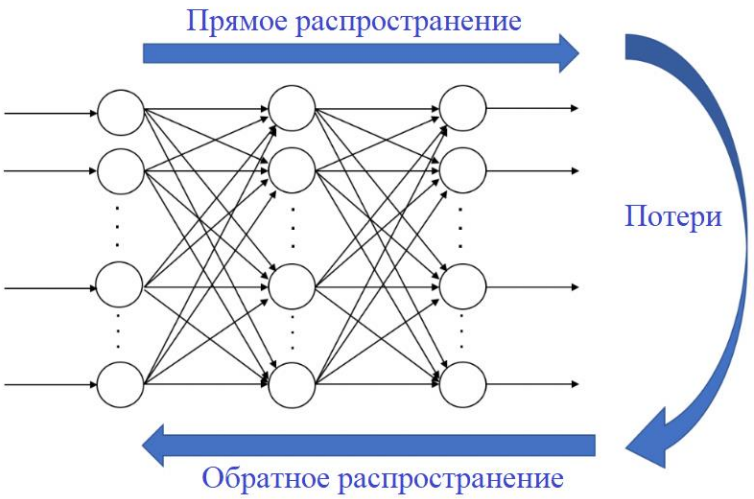

Рисунок 4 – Схема этапов обучения нейронной сети

Теперь, когда информация распространилась в обратную сторону, появляется возможность скорректировать веса связей между нейронами. Ошибка должна стать меньше в следующий раз, когда сеть будет использована для предсказания [3]. Для того чтобы осуществить это используется метод, называемый градиентным спуском. Этот метод изменяет веса с малым шагом с помощью расчета производной (или градиента) функции потерь, что позволяет увидеть, в каком направлении «спускаться» к глобальному минимуму.

Подводя итог, можно рассматривать обратное распространение как метод изменения параметров (весов и смещений) нейронной сети в правильном направлении. Сначала вычисляется значение функции потерь, а затем параметры сети корректируются при обратном проходе с использованием алгоритма оптимизации, учитывающего вычисленную ошибку [1].

Градиентный спуск является основой многих оптимизаторов и одним из наиболее распространенных алгоритмов оптимизации в машинном обучении и глубоком обучении. Для того чтобы выяснить, в каком направлении значение функции потерь уменьшается, необходимо вычислить градиент функции потерь относительно всех параметров. Градиент - это многомерное обобщение производной, это вектор, содержащий все частные

производные функции относительно каждой переменной [4]. Другими словами, это вектор, который содержит наклон функции потерь вдоль каждой оси.

Разберем пример использования метода градиентного спуска для решения задачи линейной регрессии. Формула 2 используется для вычисления среднеквадратической ошибки для некоторой задачи линейной регрессии.

$$J = \frac{1}{n}\sum_{i}(y_i - (mx_i + b)) \tag{2}$$

где n – количество элементов;

y_i – целевое значение;

x_i – полученное значение;

m и b – параметры оптимизации.

Есть два параметра, которые необходимо оптимизировать: m и b. Формулы 3 и 4 представляют частную производную функции по каждому из них.

$$\frac{\partial J}{\partial m} = \frac{2}{n}\sum_{i} x_i(y_i - (mx_i + b)), \tag{3}$$

$$\frac{\partial J}{\partial b} = \frac{2}{n}\sum_{i}(y_i - (mx_i + b)). \tag{4}$$

где n – количество элементов;

y_i – целевое значение;

x_i – полученное значение;

m и b – параметры оптимизации;

$\frac{\partial J}{\partial m}$ и $\frac{\partial J}{\partial b}$ – частные производные по параметрам.

Стоит заметить, что градиент указывает направление возрастания функции, поэтому следуют брать его отрицательным [3]. Как далеко необходимо продвинуться в полученном направлении. Это очень важный аспект. При обычном градиентном спуске размер шага остается в виде

гиперпараметра и выбирается вручную [3]. Этот гиперпараметр известен как скорость обучения - обычно является самым важным и чувствительным гиперпараметром, который нужно установить, и часто обозначается как α. Если скорость обучения установлена слишком низкой, то для достижения минимума может потребоваться недопустимо долгое время. Если скорость обучения установить слишком высокой, то минимум может быть «перепрыгнут» и ошибка может начать расти [4]. На формулах 5 и 6 представлены шаги обновления для этих двух параметров.

$$m = m - a\frac{\partial J}{\partial m}, \tag{5}$$

$$b = b - a\frac{\partial J}{\partial b}. \tag{6}$$

где m и b – параметры оптимизации;

a – гиперпараметр скорости обучения;

$\frac{\partial J}{\partial m}$ и $\frac{\partial J}{\partial b}$ – частные производные по параметрам.

Но что делать если параметров больше, чем два. Если мы обозначим все параметры как w_i, то сможем экстраполировать приведенный выше пример на многомерный случай. Это можно записать более кратко, используя градиентную нотацию. Градиент J будет обозначен как ∇J, он является вектором, содержащим частные производные. Таким образом, шаг обновления запишется как на формулах 7 и 8.

$$\nabla J(W) = \left(\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \dots, \frac{\partial J}{\partial w_N}\right), \tag{7}$$

$$W = W - a\nabla J(W). \tag{8}$$

где W – параметр оптимизации;

a – гиперпараметр скорости обучения;

$\frac{\partial J}{\partial w}$ – частные производные по параметрам;

∇J – градиент.

Вышеуказанная формула является канонической формулой для обычного градиентного спуска [4].

Разобранный пример и выкладки верны для оптимизации линейной модели, но не верны для нейронных сетей из-за нелинейности, которую вносят их функции активации. Функция потерь нейронной сети не является «чашеобразной» и выпуклой. Вместо этого ее функция потерь намного сложнее, с множеством холмов и долин, кривых и других неровностей [3]. Это означает, что существует множество локальных минимумов, то есть точек, где потери являются наименьшими в своем непосредственном окружении, но не обязательно являются глобальным минимумом всей функции. Это значит, что, если выполнить градиентный спуск, то есть вероятность случайно застрять в локальном минимуме.

Кроме локальных минимумов, обычный градиентный спуск имеет еще одну большую проблему: он слишком медленный [3]. Нейронная сеть может иметь сотни миллионов параметров. Это означает, что для оценки одного примера из обучающего набора потребуются сотни миллионов операций. Следовательно, градиентный спуск, применяемы ко всем элементам нашего набора данных буде очень затратной и медленной операцией. Есть метод, решающий как эту проблему, так и проблему локальных минимумов. Необходимо использовать модифицированную версию градиентного спуска, называемую стохастический градиентный спуск (stochastic gradient descent) или сокращенно СГС (SGD) [3]. С помощью СГС мы перемешиваем наш набор данных, а затем просматриваем каждый экземпляр по отдельности, вычисляя градиент относительно этой единственной точки, и выполняем обновление веса для всех точек. Это может показаться плохой идеей, потому что один единственный пример может оказаться отклонением и не обязательно будет давать хорошую аппроксимацию фактического градиента. Но как показывает практика, если применять это для каждого экземпляра выборки в случайном порядке, то общие колебания на пути обновления градиента усреднятся и сойдутся в хорошее решение. Более того, СГС помогает выходить из локальных минимумов и седловых точек, делая

обновления более прерывистыми и хаотичными [3]. Этого может быть достаточно, чтобы не застрять, если вдруг алгоритм зайдет в низину.

На практике СГС хорошо справляется с эффективной оптимизацией функции потерь нейронной сети. Однако у метода есть и слабые стороны. Проблема седловых точек решается не полностью. Всё еще есть шанс застрять во время выполнения алгоритма в том месте, где существует плато функции потерь и градиент становится очень близким к нулю [3]. Также скорость обучения остается гиперпараметром, который должен устанавливаться вручную перед началом обучения, что может быть затруднительным. Слишком низкий показатель скорости обучения приведет к медленной сходимости, а слишком высокий - к пропуску искомого минимума. Схема применения СГС для оптимизации функции потерь представлена на рисунке 5.

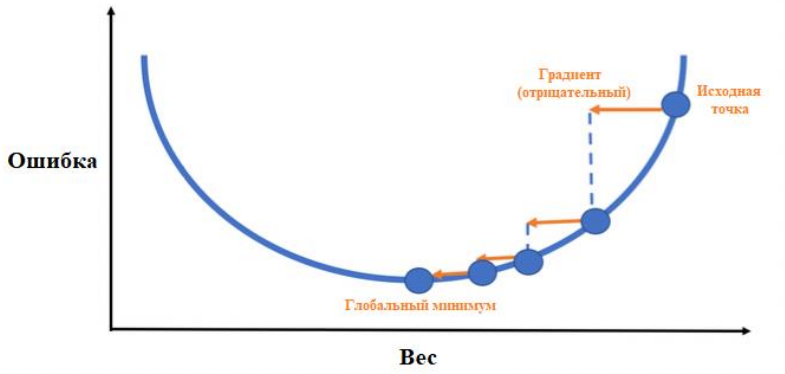

Рисунок 5 – Схема применения метода стохастического градиентного спуска

Собрав все шаги вместе получаем алгоритм обучения, который состоит из следующих основных шагов:

– задание начальных значений (часто случайных) для параметров нейронной сети (веса и смещения);

– пропускаем примеров входных данных через все слой нейронной сети для получения предсказаний;

– сравниваем полученные предсказания со значениями ожидаемых меток и рассчитываем значение функции потерь (ошибку);

– выполняем обратное распространение, чтобы распространить эту ошибку на каждый из параметров, составляющих модель нейронной сети;

– используем рассчитанные при обратном распространении поправки для корректировки параметров нейронной сети методом стохастического градиентного для того, чтобы уменьшить суммарные потери и получить более качественную модель;

– повторяем предыдущие шаги до тех пор, пока не будет получена удовлетворяющая выдвинутым требованиям модель.

На рисунке 6 представлена схема алгоритма обучения искусственной нейронной сети.

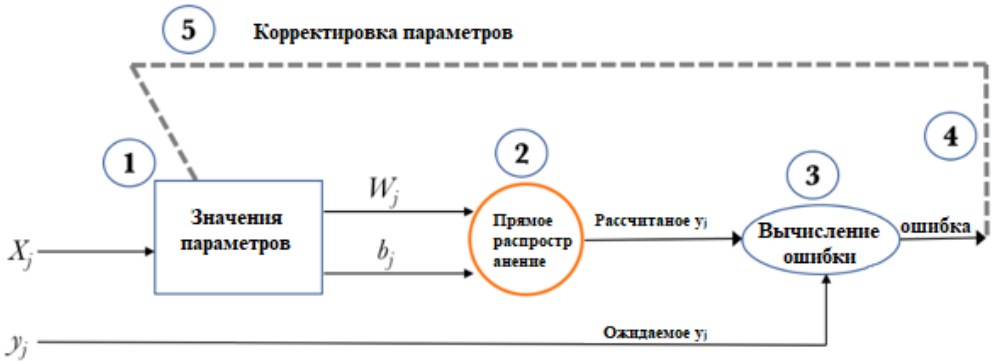

Рисунок 6 – Схема алгоритма обучения искусственной нейронной сети

Итак, в итоге был произведен анализ технологии искусственных нейронных сетей, были выявлены их особенности и свойства, разобраны составляющие их элементы и методы. Был разобран процесс построения модели нейронной сети, процесс обучения и использования. Также был рассмотрен математический аппарат и разобраны все его составляющие.

1.3 Анализ технологии свёрточных нейронных сетей

Свёрточные нейронные сети (Convolutional Neural Networks) или сокращенно СНС (CNN) очень похожи на обычные нейронные сети, разобранные в предыдущем подразделе: они состоят из нейронов, у которых

есть обучаемые веса и смещения. Каждый нейрон получает некоторые входные данные, выполняет умножение на вес и опционально применяет некоторую нелинейную функцию активации. Вся сеть по-прежнему представляет из себя функцию оценки, на входе которой - пиксели исходного изображения, а на выходе предсказания меток класса. И они все еще обладают функцией потерь, значение которой вычисляется на последнем слое, и все методы, и технологии, использующиеся для обучения обычных нейронных сетей, все еще применимы.

Тогда разберемся в чем отличие. При разработке архитектуры CHC сделано явное предположение о том, что входными данными будут изображения, что позволило заложить определенные свойства в архитектуру. Основное свойство, заложенное в архитектуру это уменьшение входного признакового пространства [5]. Это делает функцию прямого распространения более эффективной для реализации и значительно снижает количество параметров в сети.

CHC доказали свою эффективность в таких областях, как распознавание и классификация изображений. Они добились успеха в идентификации лиц, объектов и дорожных знаков, также они обеспечивают зрение для роботов и беспилотных автомобилей [15].

Обычные нейронные сети плохо масштабируются для работы с изображениями [22]. Если изображение имеет размер 32 на 32 на 3 (32 ширина, 32 высота, 3 цветовых канала), то один полносвязный нейрон в первом скрытом слое нейронной сети будет иметь 32×32×3 = 3072 значения веса. Это количество все еще кажется выполнимым, но очевидно, что эта полносвязная структура сети не масштабируется до больших изображений. Например, использование изображений более солидного размера, например, 200 на 200 на 3, приведет к тому, что нейроны будут иметь 200×200×3 = 120 000 весов. Очевидно, что такая полносвязная структура расточительна, и огромное количество параметров быстро приведет к переобучению. CHC используют тот факт, что входными данными являются изображения и они

более разумно ограничивают архитектуру [5]. В частности, в отличие от обычной нейронной сети, слои CHC состоят из нейронов, расположенных в 3-х измерениях: ширина, высота, глубина. Стоит отметить, что термин глубина здесь относится к третьему измерению объема, а не к глубине нейронной сети, который выражается в общем количестве слоев в сети. Нейроны в слое соединяются только с небольшой областью слоя, а не со всеми нейронами как в полносвязных слоях. Более того, выходной слой для первого примера с размером изображения 32 на 32 на 3 и 10 выходными классами, будет иметь размеры 1 на 1 на 10. Потому как к концу архитектуры CHC изображение сводиться в единый вектор предсказаний вероятностей классов. На рисунке 7 представлена визуализация данного процесса. Каждый слой CHC преобразует трехмерный входной массив в трехмерный выходной массив. В данном примере красный входной слой содержит изображение, поэтому его ширина и высота будут соответствовать размерам изображения, а глубина - 3 (каналы цветовой модели RGB: красный, зеленый, синий).

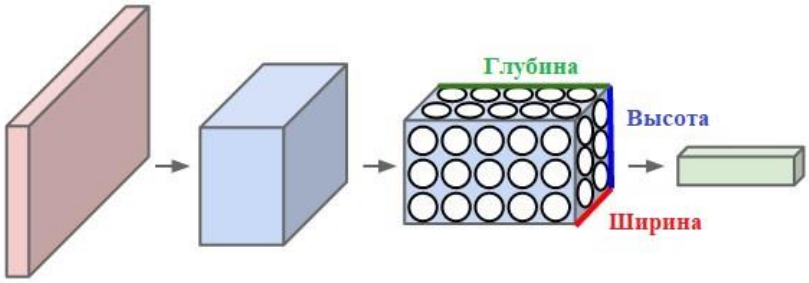

Рисунок 7 – Визуализация особенности работы CHC

Как было описано выше, простая CHC - это последовательность слоев, и каждый слой преобразует один трехмерный массив нейронов в другой с помощью дифференцируемой функции. Для построения архитектуры CHC используется три основных типа слоев: свёрточный слой (convolution layer), слой подвыборки (pooling layers) и полносвязных слоев (точно таких же, как и в обычных нейронных сетях) [5]. Объединив эти слои в стек, сформируем полную архитектуру CHC.

СНС получили свое имя от оператора свёртки. Основное назначение свёртки в случае СНС - это извлечение признаков (особенностей) из входного изображения. Операция свёртки сохраняет пространственную взаимосвязь между пикселями за счет изучения особенностей изображения с использованием небольших фрагментов (квадратов) входных данных [5]. Рассмотрим на примере.

Каждое изображение можно рассматривать как матрицу значений пикселей. Рассмотрим изображение 5 на 5, значения пикселей которого равны 0 или 1. Представим, что у нас еще одна матрица размером 3 на 3 с такими же возможными значениями элементов. Тогда свёртка изображения 5 на 5 и матрицы 3 на 3 может быть вычислена как показано на рисунке 8. Перемещаем оранжевую матрицу по исходному изображению (зеленая матрица) на 1 пиксель, называемый шагом (stride), и для каждой позиции вычисляем поэлементное умножение между двумя матрицами, затем суммируем все полученные значения, чтобы получить конечное целое число, которое образует один элемент выходной матрицы. Матрица 3 на 3 «видит» только часть входного изображения в каждом шаге. В терминологии СНС матрица 3 на 3 называется фильтром, или ядром свёртки, или детектором признаков, а матрица, сформированная путем прохода фильтра по изображению и вычисления поэлементного умножения, называется свёрнутым признаком (convolved feature), или картой активации (activation map), или картой признаков (feature map) [22]. Важно отметить, что фильтры действуют как детекторы признаков (особенностей) из исходного входного изображения. Из рисунка 8 и вышеизложенной информации становится понятно, что разные значения матрицы фильтра будут давать разные карты признаков для одного и того же входного изображения. Это означает, что различные фильтры могут обнаруживать различные особенности изображения, например, грани, кривые и так далее.

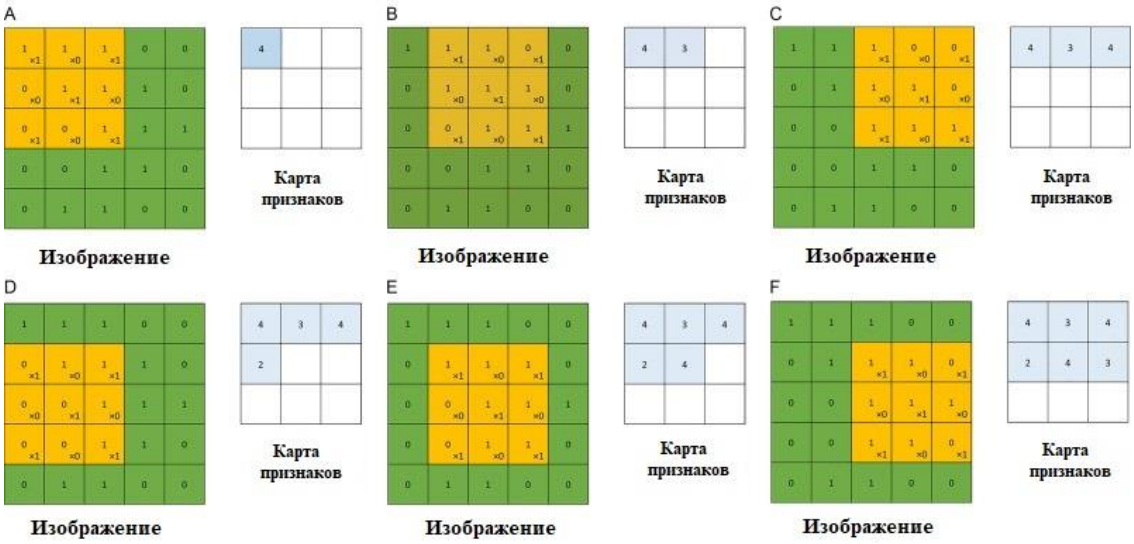

Рисунок 8 – Пример операции свёртки

На практике СНС в процессе обучения самостоятельно изучает значения фильтров, но перед обучением все равно необходимо указывать такие параметры, как количество фильтров, размер фильтра, архитектуру сети и так далее [15]. Чем больше фильтров используется, тем больше признаков изображения извлекается и тем лучше становится нейронная сеть при распознавании паттернов на неизвестных ей изображениях. Размер карты признаков контролируется тремя параметрами, которые необходимо определить до выполнения этапа свертки: глубина, шаг и дополнение нулями (zero-padding). Глубина соответствует количеству фильтров, которые используются для операции свертки. Шаг - это количество пикселей, на которое перемещается матрица фильтра по входной матрице. При большом шаге будут получаться меньшие карты признаков. Иногда бывает удобным дополнить входную матрицу нулями вокруг границы, чтобы можно было применить фильтр к граничащим элементам входной матрицы изображения [5]. Полезной особенностью дополнения нулями является то, что оно позволяет контролировать размер карт признаков. Дополнение нулями также называется широкой сверткой, а не использование его - узкой сверткой.

В свёрточных слоях для управления количеством параметров используется схема распределения параметров (parameter sharing scheme) [5].

Для обработки в свёрточный слой может поступать огромное количество параметров, что определенно приведет к большим затратам вычислительных ресурсов. Чтобы уменьшить затраты можно резко сократить количество параметров, сделав одно разумное предположение: если один признак полезен для вычисления в какой-то пространственной позиции (x,y), то он должен бать полезен она и для вычисления в другой позиции (x2,y2). Для этого разобьём трехмерный массив нейронов на срезы глубины – матрицы некоторого размера. Вычислительные затраты будут ограничены за счет того, что для всех нейронов одно среза глубины будут использованы одинаковые веса и смещение. На практике во время обратного распространения каждый нейрон в массиве вычислит градиент для своих весов, но эти градиенты будут складываться по каждому срезу глубины и обновлять только один набор весов на срез. Обратный проход при распространении для операции свертки (как для данных, так и для весов) также является сверткой, но с пространственно перевернутыми фильтрами.

Пространственное агрегирование (spatial pooling), также называемое субдискретизацией или подвыборкой, уменьшает размерность каждой карты признаков, но сохраняет наиболее важную информацию [15]. Эта операция может быть разного типа: максимальная, средняя, суммирующая и так далее. В случае максимальной подвыборки определяется пространственная окрестность (например, окно 2 на 2) и берется самый большой элемент из карты признаков в этом окне. Вместо того, чтобы взять самый большой элемент, можно взять среднее или сумму всех элементов в этом окне. На практике максимальная подвыборка показывает лучшие результаты.

На рисунке 9 представлен пример операции выполнения операции максимальной подвыборки применительно к карте признаков, полученной после свертки и. применения полулинейной функции активации (ReLU). Мы сдвигаем окно размером 2 на 2 с шагом в 2 клетки и берем максимальное значение в каждой рассматриваемой области. Это уменьшает размерность нашей карты признаков.

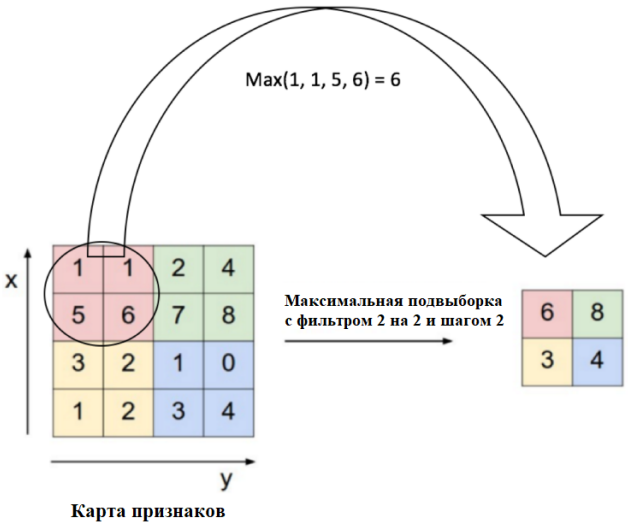

Рисунок 9 – Пример операции максимальной подвыборки

Функция подвыборки заключается в постепенном уменьшении размера входного представления [5]. В частности, она выполняет следующие задачи:

– делает входные карты признаков меньшими по размеру и более управляемыми;

– уменьшает количество параметров и вычислений в нейронной сети, таким образом, контролируя переобучение;

– делает сеть инвариантной к небольшим трансформациям и искажениям во входном изображении (небольшое искажение во входном изображении не изменит выход операции подвыборки, так как берется максимальное или среднее значение в локальном районе).

Обратный проход для операции максимум имеет простую интерпретацию, так, как только направляет градиент на вход, имеющий наибольшее значение при прямом проходе. Следовательно, при прямом проходе слоя подвыборки обычно отслеживается индекс максимального входа, чтобы маршрутизация градиента была эффективной во время обратного распространения [17].

Нейроны на полносвязном слое имеют связи со всеми нейронами на предыдущем слое, как и в обычных нейронных сетях. Выходные данные свёрточных слоев и слоев подвыборки представляют собой высокоуровневые

признаки входного изображения [5]. Назначение полносвязного слоя - использовать эти признаки для классификации входного изображения по различным классам на основе обучающего набора данных. Помимо классификации, добавление полносвязного слоя также является (обычно) дешевым способом изучения нелинейных комбинаций этих признаков. Большинство элементов из сверточного и слоев подвыборки могут быть полезны для решения задачи классификации, но комбинации этих элементов могут быть еще лучше [5].

В этом разделе была рассмотрена архитектура и принципы её построения, составные элементы, использующиеся математические операции свёрточных нейронных сетей. Также были проанализированы возможные применения СНС, рассмотрены отличия от обычных нейронных сетей и преимущества для решения поставленной задачи по распознаванию и локализации объектов на изображении.

2 Проектирование свёрточной нейронной сети для распознавания и локализации объектов на изображении

2.1 Базовая архитектура модели свёрточной нейронной сети

Для того чтобы построить модель, способную обнаруживать и локализовать определенные объекты на изображениях реализуем Single Shot Multibox Detector (SSD), популярную, мощную и особенно оперативную сеть для этой задачи. Авторами SSD являются следующие ученные и разработчики: Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. Рассмотрим эту модель, её строение, особенности и принцип работы подробнее.

Начнём с определений и терминов, которые используется в данной модели. Ограничительная рамка (bounding box) - это рамка, в которую заключается определенный объект, то есть она представляет границы этого объекта [19]. В рассматриваемой модели используются как обычные рамки, так и ограничительные. Но все рамки располагаются на изображениях, и необходимо уметь измерять их положение, формы, размеры и другие свойства. Самый очевидный способ представления рамки - по координатам пикселей линий x и y, составляющих его границы. Такие координаты называются граничными координатами (boundary coordinates). Граничные координаты рамки записываются как (x_min, y_min, x_max, y_max) [13]. Расстояние в пикселях от левого края изображения до левой и правой границы рамки это x_min и x_max соответственно, а от верхнего края до верхней и нижней границы рамки это y_min и y_max. Начало отсчета этих координат – верхний левый угол изображения. Но значения пикселей практически бесполезны, если мы не знаем фактических размеров изображения. Лучше представить все координаты в их дробной форме, то есть нормировать в диапазоне от 0 до 1 [13]. Теперь координаты не зависят от размеров изображения, и все рамки на всех изображениях измеряются в одном масштабе. Центрально-размерные координаты (Center-Size coordinates)

- это более явный способ представления положения и размеров рамки на изображении [13]. Центрально-размерные координаты рамки записываются как (c_x, c_y, w, h). Центр рамки по оси x и y - это c_x и c_y. Высота h – это длинна рамки по оси y, а ширина w - это длинна рамки по оси x. При реализации регулярно используются обе системы координат в зависимости от их пригодности для решения задачи, и всегда в их дробных формах.

Коэффициент Жаккара (Jaccard index) или наложение Жаккара (Jaccard Overlap) или пересечение над объединением (Intersection-over-Union) измеряют степень наложения двух рамок друг на друга [25]. Коэффициент Жаккара можно выразить формулой 9.

$$Jaccard\ Index = \frac{A \cap B}{A \cup B} \tag{9}$$

где A и B – два множества.

Коэффициент Жаккара равный 1 означает, что сравниваемые рамки являются одним и тем же, в то время как значение 0 означает, что они являются взаимоисключающими. Это простой показатель, но он находит множество применений в рассматриваемой модели.

Multibox - это метод обнаружения объектов, где предсказание состоит из двух компонентов [13]:

– координаты рамки, которая может содержать или не содержать объект, что является регрессионной задачей;

– оценки, описывающие вероятности присутствия в данной рамке объекта каждого из возможных классов, включая фоновый класс, который подразумевает отсутствие объекта в рамке, что является классификационной задачей.

SSD – это свёрточная нейронная сеть, которую можно представить в виде трёх частей [13]:

– базовые свёрточные слои, взятые из существующей архитектуры классификации изображений, которые обеспечат низкоуровневые карты признаков;

– дополнительные свёрточные слои, добавленные к базовым и предоставляющие высокоуровневые карты признаков;

– свёрточные слои для предсказаний, которые будут локализовать и идентифицировать объекты на этих картах признаков.

Рассмотрим базовые свёрточные слои. Прежде всего, разберемся зачем использовать свёрточные слои из существующей сетевой архитектуры. Потому что модели, которые хорошо зарекомендовали себя в задаче классификации изображений, уже довольно хорошо могут улавливать основную суть изображения [14]. Те же свёрнутые признаки будут полезны и при обнаружении объектов, хотя и в более локальном смысле - в этой задаче менее интересно распознавание целого изображения, чем конкретных его областей, в которых находятся объекты.

Есть также дополнительное преимущество - возможность использовать слои, предварительно обученные на надежном классификационном наборе данных. Этот метод называется transfer learning [24]. Заимствуя знания из другой, но тесно связанной задачи, можно добиться прогресса еще до начала обучения. Так как при transfer learning переобучаются несколько последних слоев нейронной сети, отвечающих конкретно за классификацию по заданным класса, при этом остаются обученные слои, которые отлично справляются с выделением различных особенностей и признаков объектов на изображениях.

В рассматриваемой модели SSD используют архитектуру нейронной сети VGG-16 в качестве базовой сети. Она довольно проста в своем первоначальном виде. Схема архитектуры нейронной сети VGG-16 представлена на рисунке 10. Авторы модели рекомендуют использовать нейронную сеть, которая была обучена на наборе данных из задачи по классификации из конкурса по крупномасштабному визуальному распознаванию ImageNet (ILSVRC). Также можно использовать что-то по крупнее из нейронных сетей, но стоит учитывать вычислительные затраты.

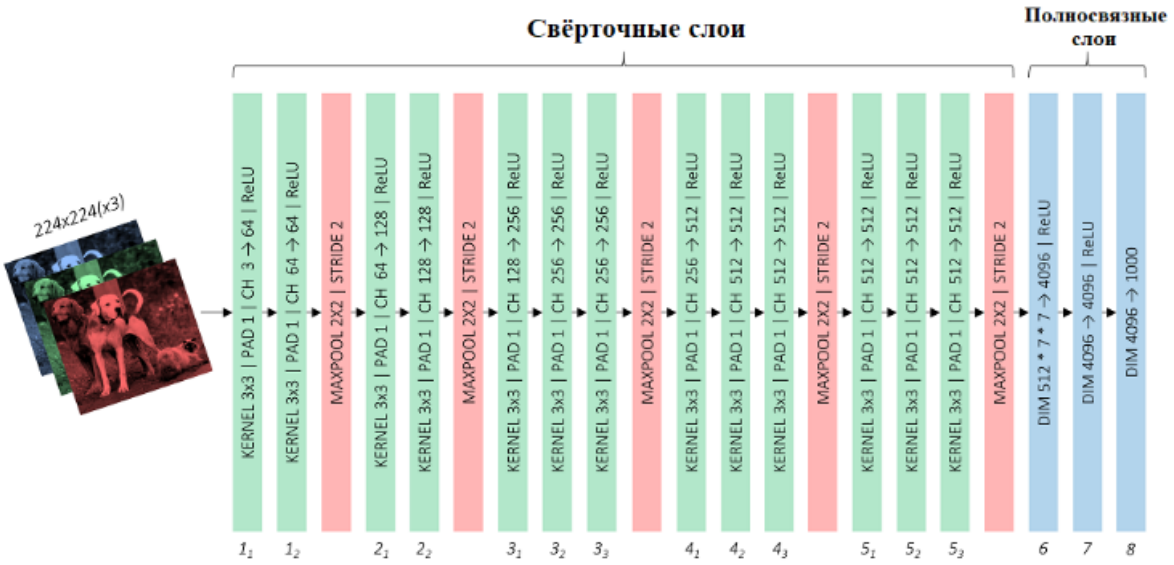

Рисунок 10 – Схема архитектуры нейронной сети VGG-16

Необходимо внести некоторые изменения в эту заранее обученную сеть, чтобы адаптировать ее к собственным задачам по обнаружению объектов [13]. Некоторые из них логичны и необходимы, в то время как другие, в основном, являются вопросом удобства или предпочтения. Итак, перейдем к этим изменениям. Размер входного изображения будет 300 на 300 пикселей. Третий слой подвыборки (pooling), который уменьшает размеры карты признаков наполовину, будет использовать математическую функцию ceiling вместо функции floor, которая стоит по умолчанию при определении выходного размера, то есть будет округлять в большую сторону. Это важно только в том случае, если размеры предыдущей карты признаков нечетные. Можно подсчитать, что для входного изображения размером 300 на 300, карта признаков на выходе свёрточного слоя, идущего перед этим слоем подвыборки, будет иметь поперечный срез 75 на 75, который в случае введённого изменения уменьшится до размеров 38 на 38, а не до неудобных 37 на 37. Также модифицируем пятый слой подвыборки с ядром свёртки 2 на 2 и шагом 2, размер ядра устанавливаем 3 на 3, а шаг 1. Это приводит к тому, что размеры карты объектов больше не уменьшаются вдвое после предыдущего свёрточного слоя. Полносвязные слои, выполняющие

классификацию, в этом случае не нужны, потому что данная сеть берётся как базовая для получения карт признаков. Отбрасываем восьмой слой полностью, но переделаем шестой и седьмой в свёрточные слои. Первые три модификации достаточно просты, но последнюю необходимо разобрать.

Рассмотрим, как переделать полносвязный слой в свёрточный. В стандартной нейронной сети для классификации изображений первый полносвязный слой не может обработать карту признаков с предыдущего слоя [13]. Необходимо преобразовать выход свёрточного слоя в одномерную структуру. Если произвести прогон некоторого изображения через полносвязный и свёрточный слой, то можно заметить, что параметры слоев будут одни и те же, отличие только структурах, в которых они хранятся. Также ключевая особенность - в обоих сценариях случаях выходы будут одинаковы, если произвести вычисления по параметрам. Можно сделать вывод, что для изображения размера H, W с I входными каналами, полносвязный слой выходного размера N эквивалентен свёрточному слою с размером ядра свёртки равному размеру изображения H, W и N выходных каналов, при условии, что параметры полносвязнной слоя совпадают с параметрами свёрточного слоя. Таким образом, любой полносвязный слой может быть преобразован в эквивалентный свёрточный слой путем простого изменения размеров структуры, хранящей эти параметры.

Теперь можно конвертировать шестой и седьмой полносвязные слои в оригинальной архитектуре VGG-16 в свёрточные. В показанной ранее архитектуре сети, которая работает на изображениях размером 224 на 224 с 3 цветовыми каналами, можно посчитать проходя по слоям и узнать, что выход с 5 свёрточного слоя будет иметь размер 7 на 7 на 512. Поэтому шестой полносвязный слой с входным размером $7 \times 7 \times 512$ и выходным размером 4096 имеет параметры 4096, $7 \times 7 \times 512$. Эквивалентный свёрточный слой будет иметь размер ядра 7 на 7 и 4096 выходных каналов с параметрами 4096, 7, 7, 512. Седьмой полносвязный слой с входом размером 4096 и выходом размером 4096 имеет параметры 4096, 4096. Входом можно считать

изображение размером 1 на 1 с 4096 входными каналами. Эквивалентный свёрточный слой имеет размер ядра свёртки 1 на 1 и 4096 выходных каналов, с параметрами 4096, 1, 1, 4096.

Видно, что в шестом сверточном слое есть 4096 фильтров, каждый с размерами 7, 7, 512, а в седьмом слое - 4096 фильтров, каждый с размерами 1, 1, 4096. Эти фильтры слишком большие и их много, вычислительные затраты на них будут велики. Для исправления этого авторы SSD архитектуры решили уменьшить как количество фильтров, так и размер каждого фильтра путем применения операции подвыборки к параметрам полученных свёрточных слоев. Шестой слой будет использовать 1024 фильтра, каждый с размерами 3 на 3 на 512. Поэтому параметры преобразуются от 4096, 7, 7, 512 до 1024, 3, 3, 512. Седьмой слой будет использовать 1024 фильтра, каждый с размерами 1 на 1 на 1024. Поэтому параметры преобразуются от 4096, 1, 1, 4096 до 1024, 1, 1, 1024. Выбор каждого m-ого параметра в определенном измерении называется децимацией (decimation). Так как к ядру свёртки шестого слоя применяется децимация с размера 7 на 7 до 3 на 3, и сохраняется только каждое 3-е значение, то теперь в ядре свёртки появляются пробелы. Поэтому необходимо сделать ядро расширенным. Этого можно добиться установлением такого параметра свёртки как расширение (dilation) в значение 3. Однако, авторы фактически используют расширение 6, потому что пятый слой подвыбокри больше не уменьшает вдвое размеры предыдущей карты признаков, так как он был изменён.

Влияние параметра расширение на операцию свёртки можно увидеть на рисунке 11. Он устанавливает расстояние между клетками ядра свёртки при его проходе по сворачиваемой карте признаков или изображению [21]. Таким образом, установив этот параметр будут пропускаться появившиеся в процессе децимации пробелы в картах признаков.

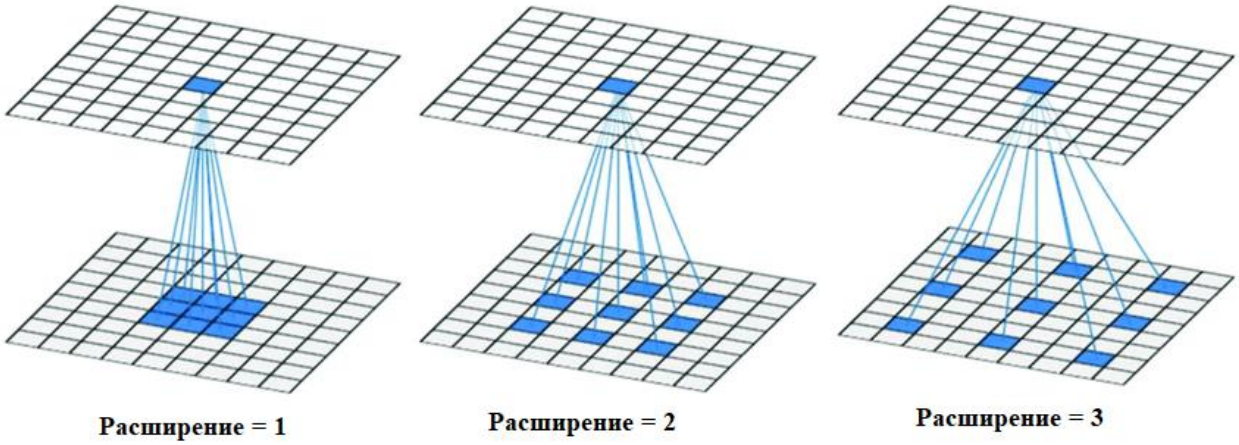

Рисунок 11 – Демонстрация влияние параметра расширение

На рисунке 12 можно увидеть получившуюся базовую сеть, а именно модифицированную VGG-16. Стоит обратить внимание на карты признаков на выходе со слоя 4_3 и 7. В дальнейшем они будут использованы в процессе локализации и распознавания.

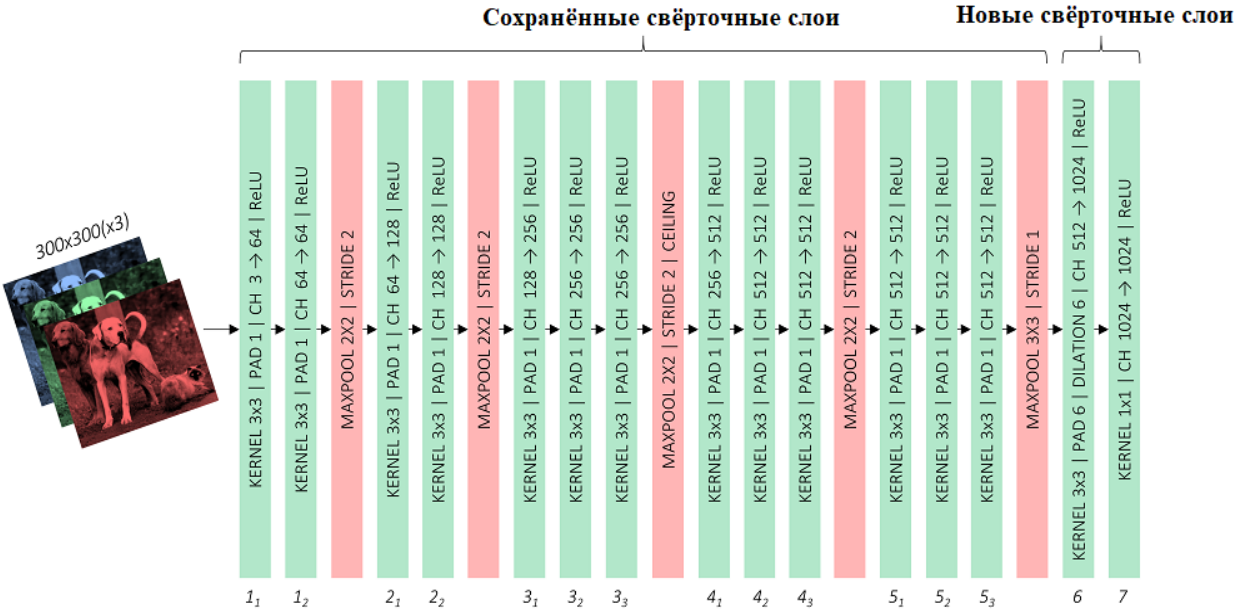

Рисунок 12 – Архитектура базовой нейронной сети

Таким образом была рассмотрена базовая архитектура используемой модели. Разобраны её слои, методы, особенности и принцип работы. Далее разберём дополнительные свёрточные слои этой архитектуры.

2.2 Дополнительные слои модели свёрточной нейронной сети

Перейдем к рассмотрению дополнительных свёрточных слоёв. Теперь добавим еще несколько свёрточных слоев поверх полученной базовой сети. Вводим в сеть четыре дополнительных свёрточных блока, каждый из которых имеет два слоя. В то время как уменьшение размера карт признаков происходило за счет операции подвыборки в базовой сети, здесь это происходит за счет установления параметра шага (stride) в 2, в каждом втором слое [13]. Эти свёрточные слои предоставляют дополнительные карты признаков, каждая из которых становится меньше по сравнению с предыдущей. Схема с дополнительными слоями отображена на рисунке 13. На входе находятся 1024 карты признаков размером 19 на 19. Далее проходя по свёрточным слоям, после каждого блока получаем очередную карту признаков, которая будет использоваться в процессе распознавания объектов.

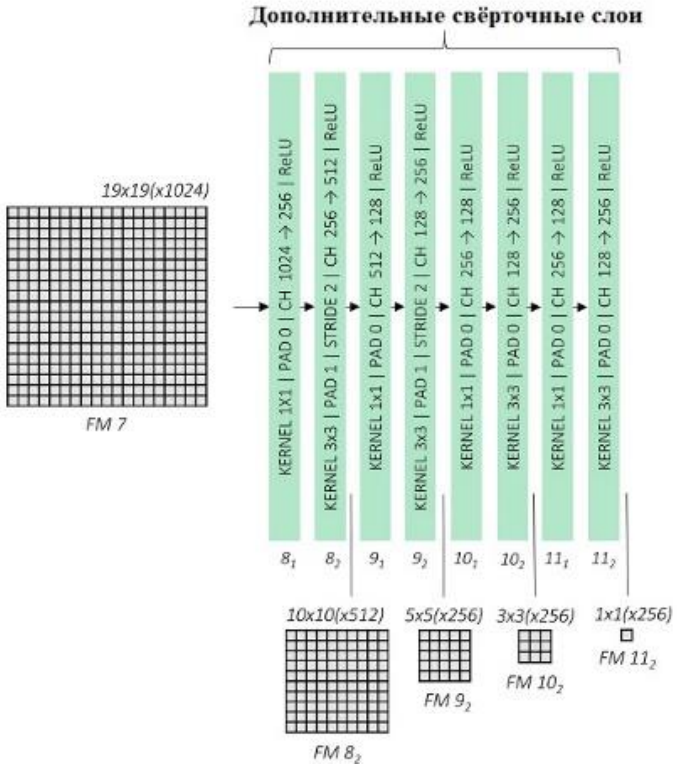

Рисунок 13 – Архитектура дополнительных свёрточных слоев

Остановимся на происходящих здесь преобразованиях по подробнее. В слоях используются трехмерные свёртки (3d convolution). Термин ядро свёртки относится к двухмерному массиву весов. В то время как термин фильтр используется для трехмерных структур, состоящих из нескольких ядер, сложенных вместе. Для двухмерных фильтров эти термины обозначают одно и тоже. Но для трёхмерного фильтра, который здесь используется, фильтр - это набор из ядер свёртки [6]. Каждое ядро уникально, и оно подчеркивает различные аспекты входного канала. С этими понятиями трёхмерная свертка происходит следующим образом. Каждое ядро накладывается на входной канал слоя для создания одной карты признаков выходного канала. Мы повторяем этот процесс для всех ядер, чтобы сгенерировать несколько каналов [6]. Каждый из этих каналов затем суммируется, образуя один единственный выходной канал.

Допустим входной слой представляет собой матрицу 5 на 5 с 3 каналами. Фильтр представляет собой массив размером 3 на 3 на 3. Сначала каждое из ядер в фильтре применяется к трем каналам на входном слое, по отдельности. Выполняются три свертки, в результате чего получаются 3 канала размером 3 на 3. Затем эти три полученных канала суммируются вместе (поэлементное сложение) и образуют один единственный канал (3 на 3 на 1). Этот канал является результатом свертки входного слоя (5 на 5 с 3 каналами) с помощью фильтра (3 на 3 на 3). Полученные после свертки три канала, а также их сложение в выходной канал отображено на рисунке 14. Можно представить этот процесс как скольжение матрицы трёхмерного фильтра по входному слою. Глубина слоя и фильтра должны совпадать.

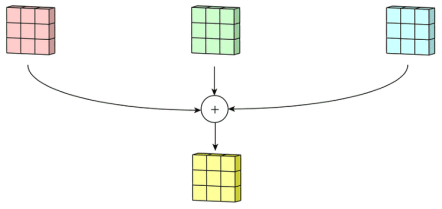

Рисунок 14 – Схема трёхмерной свёртки

Можно совершать переходы между слоями с разной глубиной [6]. Допустим, у входного слоя есть Din каналов, и есть необходимость чтобы у выходного слоя было Dout каналов. Для этого нужно просто применить Dout фильтров ко входному слою. Каждый фильтр будет иметь по Din ядер, то есть его глубина будет равна глубине входного слоя. Каждый фильтр предоставляет один выходной канал. После применения Dout фильтров получаем Dout каналов, которые затем можно сложить вместе для формирования выходного слоя. Схема описанного процесса представлена на рисунке 15.

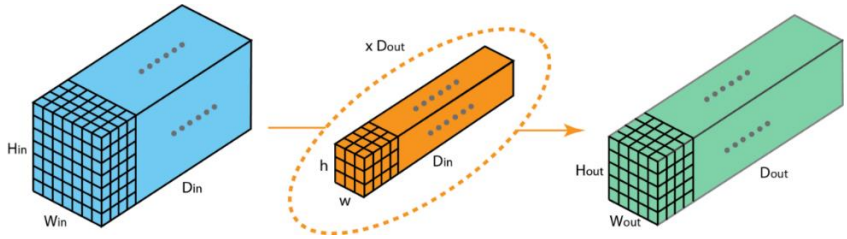

Рисунок 15 – Схема применения трёхмерной свёртки

В дополнительных свёрточных слоях используется фильтр, ядра которого имеют размер 1 на 1. Разберемся почему использование таких ядер полезно. Эти ядра тривиальны для слоев с одним каналом. В них просто умножается каждый элемент на число. Всё становятся интересным, если входной слой имеет несколько каналов. На рисунке 16 показано, как работает свертка 1 на 1 для входного слоя с размером H x W x D. После свертки 1 на 1 с размером фильтра 1 x 1 x D размер выходного канала будет H x W x 1. Если применить N таких сверток 1 на 1, а затем сложить их, то получится выходной слой размером H x W x N. С помощью данной свёртки в дополнительных свёрточных слоях происходит сокращение количества каналов для более эффективных вычислений, при этом размер карт признаков остаётся прежним [6]. Также стоит отметить, что каждый второй слой помимо размера шага 2, который используется вместо операции подвыборки для уменьшения размера карт признаков, также используется

параметр дополнения нулями со значением 1 для того, чтобы получить необходимый размер выходных карт.

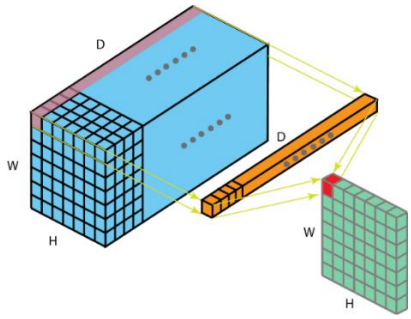

Рисунок 16 – Схема применения трёхмерной свёртки размером 1 на 1

Опять же, обращаем внимание на карты признаков, полученные на выходе свёрточных слоев 8_2, 9_2, 10_2 и 11_2. Они также пригодятся для распознавания и локализации объектов [13].

Прежде чем перейти к свёрточным слоям для предсказаний, необходимо сначала понять, что сеть будет предсказывать. Конечно, это объекты и их позиции на изображении, но в какой форме. Необходимо разобрать такой элемент как рамки по умолчанию (default boxes), также называемые приорами (priors) и их решающую роль, которую они играют в модели SSD. Предсказания объектов могут быть весьма разнообразны, и это относится не только к их типу. Они могут возникать в любом месте изображения, с любым размером и формой. Есть бесконечное число возможностей, где и как может появиться объект [25]. Хотя это утверждение может быть верно с математической точки зрения, многие варианты их расположения просто маловероятны или неинтересны. Более того, нет необходимости настаивать на том, чтобы рамки были идеально точными. По сути, можно дискретизировать математическое пространство потенциальных предсказаний всего лишь на тысячи возможностей. Приоры - это предварительно рассчитанные, фиксированные рамки, которые в совокупности представляют это множество вероятных и приблизительных предсказаний рамок [13]. Приоры выбираются вручную, но тщательно,

исходя из форм и размеров рамок объектов из обучающего набора данных (которые называют ground truth objects). Размещая эти приоры во всех возможных местах карты признаков, также учитывается разнообразие в местоположении [13].

При определении приоров авторы архитектуры уточняют, что они будут применены к различным низкоуровневым и высокоуровневым картам признаков, а именно к картам, полученным после свёрточных слоёв 4_3, 7, 8_2, 9_2, 10_2 и 11_2. Это те самые карты признаков, которые были отмечены ранее и изображены на рисунках. Если масштаб приора равен s, то его площадь равна площади квадрата со стороной s. Самая большая карта признаков, со свёрточного слоя 4_3, будет содержать приоры с масштабом 0.1, то есть 10% от размеров изображения, в то время как остальные карты признаков будут содержать приоры с масштабом, линейно увеличивающимся от 0.2 до 0.9. Как видно, большие карты признаков содержат приоры с меньшим масштабом и поэтому идеально подходят для обнаружения небольших объектов. Для каждой позиции на карте признаков будут присутствовать приоры с различными соотношениями сторон. На всех карта признаков будут приоры с соотношениями сторон 1:1, 2:1, 1:2. Промежуточные карты признаков из свёрточных слоёв 7, 8_2 и 9_2 также будут иметь приоры с соотношениями сторон 3:1, 1:3. Кроме того, все карты признаков будут иметь один дополнительный приор с соотношением сторон 1:1 и масштабом, который является средним геометрическим от масштаба текущей и последующей карты признаков. В таблице 1 представлены обобщенные данные о размещение и количестве приоров. Всего в SSD для изображений размером 300 на 300 определено 8732 приора [13].

Были определены приоры с точки зрения их масштабов и соотношений сторон, выразить это можно через формулы 10 и 11.

$$w \cdot h = s^2, \tag{10}$$

$$\frac{w}{h} = a. \tag{11}$$

где w и h – ширина и высота приора;

s и a – это масштаб и соотношение сторон.

Решение этих уравнений представлено на формулах 12 и 13.

$$w = s \cdot \sqrt{a}, \tag{12}$$

$$h = \frac{s}{\sqrt{a}}. \tag{13}$$

где w и h – ширина и высота приора;

s и a – это масштаб и соотношение сторон.

Теперь есть возможность нарисовать их на соответствующих картах признаков.

Таблица 1 – Обобщение данных о размещении и количестве приоров

Слой карты признаков	Размеры карты признаков	Масштаб приоров	Соотношения сторон	Количество приоров на позицию	Общее количество приоров на карту признаков
Слой 4_3	38 на 38	0.1	1:1, 2:1, 1:2 + экстра приор	4	5776
Слой 7	19 на 19	0.2	1:1, 2:1, 1:2, 3:1, 1:3 + экстра приор	6	2166
Слой 8_2	10 на 10	0.375	1:1, 2:1, 1:2, 3:1, 1:3 + экстра приор	6	600
Слой 9_2	5 на 5	0.55	1:1, 2:1, 1:2, 3:1, 1:3 + экстра приор	6	150
Слой 10_2	3 на 3	0.725	1:1, 2:1, 1:2 + экстра приор	4	36
Слой 11_2	1 на 1	0.9	1:1, 2:1, 1:2 + экстра приор	4	4
Всего	–	–	–	–	8732

Например, попробуем визуализировать, как будут выглядеть приоры на центральной ячейке карты признаков из свёрточного слоя 9_2. Результат

представлен на рисунке 17. Для данной ячейки есть 5 приоров с соотношением сторон 1:1, 2:1, 1:2, 3:1, 1:3 и площадью 0.55, и 1 дополнительный приор с соотношением 1:1 и площадью 0.63. Те же самые приоры существуют и для каждой из других ячеек. Если приор какой-то из ячеек выходит за края карты признаков, то он обрезается.

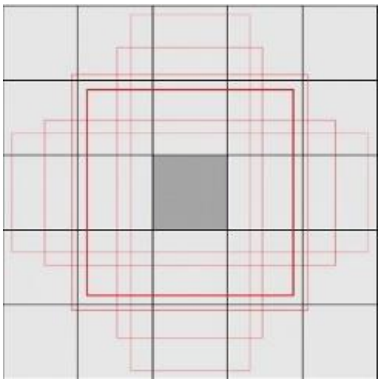

Рисунок 17 – Схема наложения приоров на карту признаков

Ранее было сказано, что регрессия будет использоваться для того, чтобы найти координаты ограничивающей объект рамки. Но тогда, появляется вопрос, могут ли приоры представлять окончательные предсказанные нейронной сетью рамки объектов. Они не могут. Приоры представляют собой приблизительные возможности для итогового предсказания [25]. Это означает, что каждый приор используется в качестве приблизительной отправной точки, а затем выясняется, насколько он нуждается в корректировке для того, чтобы получить более точное предсказание для ограничивающей рамки объекта. Поэтому, если каждая предсказываемая ограничивающая рамка - это небольшое отклонение от приора, и целью является вычисление этого отклонения, то необходимо найти способ для измерения или количественной оценки отклонения.

Рассмотрим некоторый объект на изображении, его предсказанную ограничивающую рамку, и приор, по которому было сделано предсказание. Предположим, что они представлены в центрально-размерных координатах, которые были представлены ранее. Для примера возьмем изображение, представленное на рисунке 18. Расположение и размер ограничивающих

рамок также могут быть представлены в виде набора отклонений $(g_{cx}, g_{cy}, g_w, g_h)$ от приора. Этот набор отклонений описывает насколько приор должен быть скорректирован, чтобы получить ограничивающую рамку.

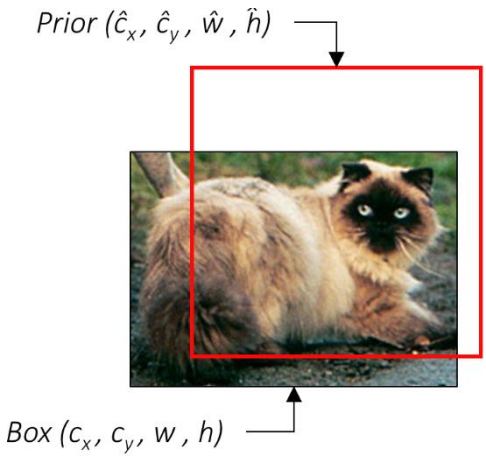

Рисунок 18 – Изображение объекта с наложенным приором

Отклонения вычисляются по формулам 14, 15, 16 и 17.

$$g_{cx} = \frac{c_x - \hat{c}_x}{\hat{w}}, \tag{14}$$

$$g_{cy} = \frac{c_y - \hat{c}_y}{\hat{h}}, \tag{15}$$

$$g_w = \ln\left(\frac{w}{\hat{w}}\right), \tag{16}$$

$$g_h = \ln\left(\frac{h}{\hat{h}}\right). \tag{17}$$

где g_{cx}, g_{cy}, g_w, g_h – отклонения рамки от приора;

$\hat{c}_x, \hat{c}_y, \hat{w}, \hat{h}$ – координаты приора;

c_x, c_y, w, h - координаты рамки.

Учитывая, что каждый приор корректируется для получения более точного предсказания, этот набор отклонений $(g_{cx}, g_{cy}, g_w, g_h)$ является той формой, в которой будет производиться регрессия координат ограничивающих рамок. Как видно, каждое отклонение нормируется соответствующим размером приора. Это имеет смысл, потому что

определенное отклонение было бы менее значительным для более крупного приора, чем для меньшего приора.

По итогам подраздела были разобраны дополнительные слои модели, их строение и особенности, выполняемая ими функция. Также был рассмотрена технология приоров. В следующем подразделе будут рассмотрены последние завершающие слои модели SSD.

2.3 Получение предсказаний модели свёрточной нейронной сети

Теперь перейдем к рассмотрению свёрточных слоёв, которые непосредственно используется для предсказаний ограничивающих рамок и классов объектов в них. Ранее мы выделили и определили приоры для шести карт признаков различного масштаба, а именно для карт признаков из свёрточных слоёв 4_3, 7, 8_2, 9_2, 10_2, и 11_2. Для каждого приора в каждой ячейке на каждой карте признаков, необходимо предсказать:

– набор отклонений $(g_{cx}, g_{cy}, g_w, g_h)$ для ограничивающей рамки;

– набор из n оценок для ограничивающей рамки, где n – это общее количество типов объектов (включая фоновый класс).

Для того, чтобы сделать это как можно проще, понадобятся два свёрточных слоя для каждой карты признаков. Свёрточный слой для предсказания локализации с размером ядра свёртки 3 на 3, который применяется на каждой ячейке карты признаков, с параметрами шага и добавления нулей 1, с 4 фильтрами для каждого приора данной ячейки [13]. Четыре фильтра для приора вычисляют четыре отклонения $(g_{cx}, g_{cy}, g_w, g_h)$ для ограничивающей рамки, предсказанной из этого приора. Свёрточный слой для предсказания классов с ядром 3 на 3, также с шагом и добавлением нулей 1, с количеством фильтров равным числу классов объектов для каждого приора ячейки. Фильтры для каждого приора вычисляют набор оценок для всех классов этого приора. Схема с картами признаков и этими свёрточными слоями представлена на рисунке 19. На выходе свёрточного

слоя слева количество каналов будет равно количеству приоров на ячейку карты признаков умноженному на 4, так как у ограничивающей рамки 4 отклонения. На выходе правого слоя количество каналов вычисляется как количество приоров на ячейку карты признаков умноженное на количество классов объектов.

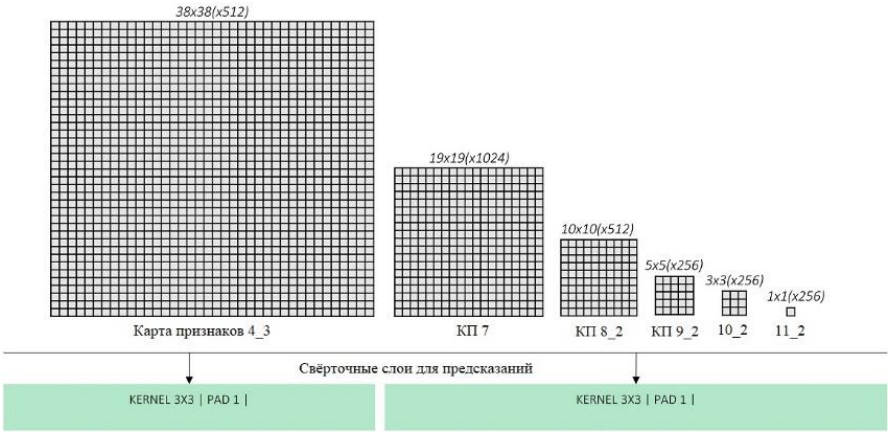

Рисунок 19 – Схема с картами признаков и слоями, применяемыми к ним

Все фильтры применяются с размером ядра 3 на 3. На самом деле не необязательно чтобы размер ядер (или фильтров) был такой же, как и у приоров, потому что различные фильтры обучаться делать предсказания относительно различных форм приоров [13]. Рассмотрим выходы этих свёрток на примере карты признаков из слоя 9_2. Процесс применения свёрток отображен на рисунке 20. Выходы слоёв локализации и слоёв предсказания классов показаны синим и жёлтым цветом соответственно. Видно, что размеры поперечного сечения 5 на 5 остаются неизменными. Что действительно интересует в этих выходах, так это третье измерение, то есть количество выходных каналов. В них хранятся фактические предсказания. Если выбрать любую из ячеек в предсказаниях для локализации и развернуть её, то можно увидеть, что это 24 канала, которые хранят 6 предсказаний рамок объектов для данной ячейки, то есть 6 наборов отклонений $(g_{cx}, g_{cy}, g_w, g_h)$ для 6 приоров. Значения каналов в каждой ячейки предсказания локализации представляют собой наборы отклонений относительно приоров для этих ячеек. Теперь сделаем то же самое с

предсказаниями классов. Допустим количество классов равно 3. Получим карту признаков размером 5 на 5 и глубиной 18 каналов. При развороте одной из ячеек получим 18 каналов, которые содержат 6 наборов по 3 оценки класса для 6 приоров этой ячейки. Эти каналы представляют собой оценки классов для приоров определенной ячейки.

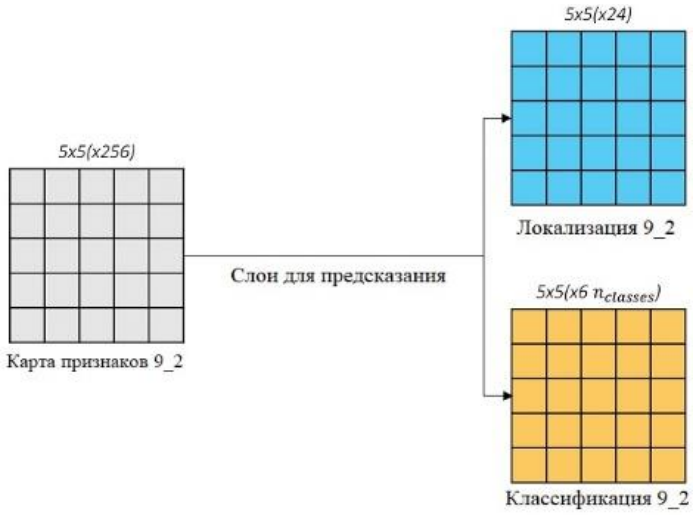

Рисунок 20 – Схема применения свёрточных слоёв для предсказаний

Теперь, зная, как выглядят предсказания для карты признаков слоя 9_2, можно преобразовать их в более удобную для обработки форму. Разворачиваем ячейки таким образом, чтобы получить общую матрицу размером: для локализации 150 строк и 4 столбца, для классификации 150 строк и 3 столбца. Каждая строка в этих матрицах хранит одно из предсказаний ограничивающей рамки и классов в ней. 150 предсказаний выставлены последовательно друг за другом. Для человеческого разума это представление выглядеть более понятно. Не будем останавливаться на этом. Можно сделать то же самое для предсказаний со всех слоев и сложить их вместе. Ранее был вычислено, что для модели определено 8732 приора. Таким образом, будет 8732 предсказанные рамки в виде наборов смещений, и 8732 набора оценок классов. Это конечный результат этапа пресдказания. Стек рамок и оценок того, что в них.

Основываясь на природе выдаваемых моделью предсказаний, легко понять, зачем может понадобиться уникальная функция потерь. Часто в

алгоритмах происходят вычисление потерь для регрессии или классификации, но довольно редко, если вообще когда-либо, для обеих задач вместе [25]. Очевидно, что общие потери должны быть совокупностью потерь от обоих типов предсказаний - локализации ограничивающих рамок и оценок классов.

Суть любого алгоритма обучения с учителем заключается в том, что должна быть возможность сопоставлять предсказания с их размеченными примерами из обучающего набора данных [11]. Здесь всё немного сложнее, так как обнаружение объектов является более открытой задачей, чем обычная задача обучения. Для того, чтобы модель чему-либо научилась, нужно структурировать задачу таким образом, чтобы можно было сравнивать предсказания с объектами, реально присутствующими на изображении. Приоры позволяют сделать именно это.

Рассмотрим алгоритм, по которому сопоставляются реальные объекты из размеченных изображений с предсказаниями этих объектов. Начинаем с того, что находим коэффициент наложения Жаккара между 8732 приорами и N объектами из выборки. Это будет тензор размера 8732, N. Затем сопоставляем каждый из 8732 приоров с объектом, с которым он имеет наибольшее наложение друг на друга. Если приор совпадает с объектом с наложением Жаккара менее 0.5, то нельзя сказать, что он содержит этот объект, и, следовательно, является отрицательным совпадением. Если учесть, что у в архитектуре используются тысячи приоров, то большинство из них будут определены при сравнении с объектом как отрицательные совпадения. С другой стороны, небольшое количество приоров на самом деле будет значительно накладываться (больше, чем 0.5) на объект, и можно сказать, что они содержат этот объект. Их называют положительными совпадениями. Теперь, когда мы сопоставили каждый из 8732 приоров с реальными объектами, по сути, также сопоставили соответствующие 8732 предсказания с этими же объектами, что и было необходимо сделать.

Рассмотрим этот алгоритм на примере. Допустим имеется некоторое изображение (рисунок 21) с наложенными на него приорами и выделенными на нём реальными объектами. Для удобства предположим, что всего имеется семь приоров, выделенных красным цветом. Реальные объекты- желтого цвета, на этом изображении три реальных объекта.

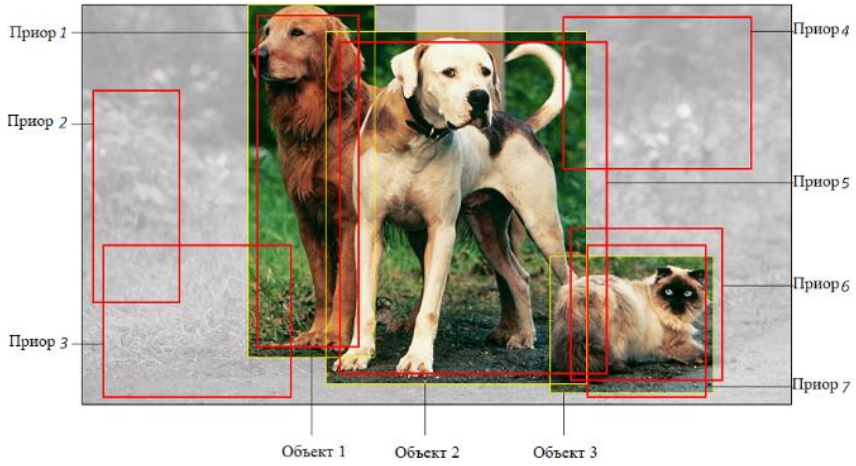

Рисунок 21 – Изображение с приорами и выделенными объектами

Следуя вышеизложенным шагам, получим следующие таблицы совпадений (таблица 2 и таблица 3).

Таблица 2 – Коэффициенты Жаккра для совпадений приоров и объектов

IoU	**Объект 1**	**Объект 2**	**Объект 3**
Приор 1	0.75	0.07	0
Приор 2	0	0	0
Приор 3	0.05	0	0
Приор 4	0	0.03	0
Приор 5	0.05	0.88	0.06
Приор 6	0	0.03	0.65
Приор 7	0	0	0.68

Таблица 3 – Совпадения приоров и объектов

Приоры	**Совпавшие объекты**	**Тип совпадения**	**Метка класса**	**Координаты**
Приор 1	Объект 1	Положительное	Собака	Объект 1

Приоры	Совпавшие объекты	Тип совпадения	Метка класса	Координаты
Приор 2	Объект 1	Отрицательное	Фон	-
Приор 3	Объект 1	Отрицательное	Фон	-
Приор 4	Объект 2	Отрицательное	Фон	-
Приор 5	Объект 2	Положительное	Собака	Объект 2
Приор 6	Объект 3	Положительное	Кошка	Объект 3
Приор 7	Объект 3	Положительное	Кошка	Объект 3

Итак, у каждого приора есть совпадение. Таким образом, каждое предсказание имеет совпадение, положительное или отрицательное. Предсказания, которые положительно сопоставляются с объектом, теперь имеют координаты реального объекта, которые будут выполнять роль целевых значений для локализации, то есть использоваться в регрессионной задаче. Естественно, что при отрицательном совпадении целевых координат нет. Все предсказания сопоставлены с метками классов реальных объектов, которая является либо типом объекта при положительном совпадении, либо фоновым классом при отрицательном совпадении. Они используются в качестве целевых меток для предсказания классов, то есть в задаче классификации.

Итак, для отрицательных совпадений отсутствуют координаты реальных объектов, которые могли бы использоваться в качестве целевых. И в этом есть смысл. Зачем обучать модель рисованию рамок вокруг пустых пространств. Поэтому потери локализации вычисляются только по тому, насколько точно была произведена регрессия позитивно совпавших предсказанных рамок [13]. Так как мы предсказывали локализацию рамок в виде смещений $(g_{cx}, g_{cy}, g_w, g_h)$, то перед вычислением потерь также нужно будет соответствующим образом преобразовать координаты реальных объектов. Потери локализации - это среднее от значений функции потерь Smooth L1 между смещениями положительно совпавших рамок и соответствующих координат реальных объектов [13]. Вычисление потерь

локализации производится по формуле 18, а вычисление функции потерь Smooth L1 по формуле 19. В формуле 19 a – это гиперпараметр и обычно берётся значение 1 [21].

$$L_{loc} = \frac{1}{n_{positives}}\left(\sum_{positives} Smooth\ L1\ Loss\right) \quad (18)$$

где L_{loc} – потери локализации;

$n_{positives}$ – количество положительно совпавших рамок;

$Smooth\ L1\ Loss$ – функция потерь.

$$Smooth\ L1(x) = \begin{cases} |x| & |x| > a \\ \frac{1}{|a|}x^2 & |x| \leq a \end{cases} \quad (19)$$

где x – переменная;

a – это гиперпараметр.

Smooth L1 Loss сочетает в себе преимущества L1-loss (устойчивые градиенты для больших значений x) и L2-loss (меньшие колебания при обновлении при малом x) [16].

У каждого предсказания, независимо от того, положительное оно или отрицательное, есть связанная с ним метка класса объекта. Важно, чтобы модель могла распознавать как объекты, так и их отсутствие. Однако, учитывая, что обычно на изображении присутствует лишь небольшое количество объектов, а подавляющее большинство из тысяч предсказаний, которые делаются, на самом деле не содержат объекта. Если отрицательных совпадений становится подавляющее большинство, то в итоге получиться модель, которая с меньшей вероятностью обнаруживает объекты, потому что в основном она обучалась обнаружению фонового класса. Решение может быть очевидным - ограничить количество отрицательных совпадений, которые будут оцениваться при расчете функции потерь. Но какие из них выбрать. Почему бы не использовать те, в которых модель больше всего ошибалась. Другими словами, использовать только те предсказания, в

которых модели сложнее всего распознать отсутствие объектов. Этот метод называется Hard Negative Mining [13].

Количество сильно отрицательных примеров, которое будет использоваться обозначим как N_hn. Это число обычно является фиксированным и кратным количеству положительных совпадений для этого изображения. В данном конкретном случае авторы архитектуры SSD рекомендуют устанавливать число сильно негативных примеров в три раза больше чем число позитивных, то есть N_hn = 3 * N_p. Наиболее сильно отрицательные примеры обнаруживаются путем нахождения потерь с помощью перекрестной энтропии (Cross Entropy) для каждого отрицательно совпавшего предсказания и выбора тех, у которых самые высокие значения [24]. Тогда потеря классификации вычисляется как сумма потерь перекрестной энтропии положительных и отрицательных совпадений. Также стоит заметить, что потери усредняются по количеству положительных совпадений. Формула 20 используется для вычисления потерь классификации.

$$L_{conf} = \frac{1}{n_{positives}}\left(\sum_{positives} CE\ Loss + \sum_{hard\ negatives} CE\ Loss\right) \quad (20)$$

где L_{conf} – потери классификации;

$n_{positives}$ – количество положительно совпавших рамок;

$CE\ Loss$ – функция потерь перекрестной энтропии.

Multibox потери представляют собой совокупность двух рассмотренных потерь, объединенных в соотношении α, представленных формулой 21.

$$L = L_{conf} + a \cdot L_{loc} \quad (21)$$

где L_{conf} – потери классификации;

L_{loc} – потери локализации;

a – параметр соотношения.

В общем случае, нет необходимости выбирать значение для α. Это может быть изучаемый параметр [13]. Для архитектуры SSD, однако, авторы просто используют α = 1, то есть производят сложение двух потерь.

После того, как модель обучена, можно применять ее к изображениям. Однако, предсказания все еще находятся в необработанном виде - два тензора, содержащих смещения и оценки классов для 8732 приоров. Они должны быть обработаны для получения окончательных, понятных для человека ограничивающих рамок с метками классов. Для это необходимо выполнить ряд действий. После выполнения прямого прохода имеется 8732 предсказанных рамок, представленных в виде набора смещений $(g_{cx}, g_{cy}, g_w, g_h)$ от соответствующих им приоров. Преобразуем их обратно до граничных координат, которые лучше интерпретируются. Затем, для каждого не фонового класса выполним следующее:

– извлекаем оценки соответствующее этому классу из всех 8732 рамок;

– удаляем рамки, которые не соответствуют определенному порогу для этих оценок (например, 0.5);

– оставшиеся (не удалённые) рамки являются кандидатами на этот конкретный класс объекта.

На данном этапе, если нарисовать все эти рамки-кандидаты на оригинальном изображении, то увидим много сильно перекрывающих друг друга рамок, многие из которых явно избыточны [13]. Это происходит потому, что очень вероятно, что из тысяч приоров, имеющихся в распоряжении, более одного предсказания соответствует одному и тому же объекту. Для человека может быть очевидным, какие рамки относятся к одному и тому же объекту. Это связано с тем, что человеческий разум может понять, что некоторые рамки в значительной степени совпадают друг с другом и с определенным объектом. Как сделать тоже самое, но применительно к компьютеру. Во-первых, выстраиваем кандидатов для каждого класса с точки зрения их вероятности, то есть сортируем их по

оценкам. Следующий шаг - выяснить, какие кандидаты лишние. Для этого используем знакомый инструмент, позволяющий судить о том, сколько общего между двумя рамками – коэффициент Жаккара. Таким образом, если найти схожесть между всеми кандидатами в данном классе, можно оценить каждую пару и, если обнаружить, что они значительно пересекаются, оставить только более вероятного кандидата. Этот процесс называется не максимальное подавление (Non-Maximum Suppression), потому что, когда обнаруживается, что несколько кандидатов значительно пересекаются друг с другом настолько, что они могут ссылаться на один и тот же объект, производим подавление всех, кроме одного, с максимальной оценкой. Алгоритмически это осуществляется следующим образом:

– располагаем кандидатов для определенного класса в порядке уменьшения оценки;

– допускаем что взяли кандидата с наивысшей оценкой и устраняем всех кандидатов с меньшей оценкой, у которых коэффициент наложения Жаккара больше, чем 0.5 с рассматриваемым кандидатом;

– допускаем, что кандидат с большей оценкой всё еще остался в пуле и исключаем из списка всех кандидатов с меньшей оценкой, у которых показатель наложения Жаккара составляет более 0.5 с рассматриваемым кандидатом;

– повторяем до тех пор, пока не пройдем всю последовательность кандидатов.

В итоге получится только одна самая подходящая рамка для каждого объекта на изображении. Не максимальное подавление имеет решающее значение для получения качественных распознаваний. Также это последний шаг для получения понятного предсказания.

В этом разделе была разобрана и проанализирована структура свёрточной нейронной сети модели SSD. Рассмотрены особенности строения слоёв, входные и выходные данные, процесс предсказания, механизм работы

трёхмерных свёрток, принцип работы приоров, вычисление потерь multibox, построена архитектура сети и внесены соответствующие коррективы. Также были разобраны дополнительные методы и функции, использующиеся в данной модели. Таким образом, была спроектирована свёрточная нейронная сеть для локализации и распознавания объектов на изображениях, к реализации которой перейдем в следующем разделе.

3 Реализация и тестирование свёрточной нейронной сети для локализации и распознавания объектов на изображениях

3.1 Реализация свёрточной нейронной сети

Для реализации модели свёрточной нейронной сети SSD, разобранной в предыдущем разделе был выбран язык программирования Python. Компания Python довольно быстро становится лучшим выбором среди разработчиков систем искусственного интеллекта, машинного обучения и проектов глубокого обучения. Разберемся почему этот язык хорошо подходит для реализации проектов с искусственным интеллектом. Одним из аспектов, делающих Python таким популярным выбором, является обилие библиотек и фреймворков, облегчающих кодирование и экономящих время на разработку. Библиотека NumPy, используемая для научных вычислений, SciPy для продвинутых вычислений и Scikit-learn для добычи и анализа данных, являются одними из самых популярных библиотек, работающей вместе с такими мощными фреймворками, как TensorFlow, CNTK и Apache Spark [7]. С точки зрения машинного и глубокого обучения, эти библиотеки и фреймворки по сути являются ориентирована на Python в первую очередь, в то время как некоторые, такие как фреймворк PyTorch, разработаны специально для Python.

Python известен своим лаконичным, читабельным кодом и практически не имеет себе равных, когда речь заходит об удобстве и простоте использования. Это имеет ряд преимуществ для машинного обучения и глубокого обучения. Обе технологии полагаются на чрезвычайно сложные алгоритмы и многоступенчатые рабочие процессы, поэтому используя понятный код и дополнительные библиотеки Python разработчик может сосредоточиться на поиске решений проблем и достижении целей проекта [8]. Простой синтаксис Python означает, что он также быстрее в разработке, чем многие языки программирования, и позволяет разработчику быстро тестировать разрабатываемые алгоритмы без необходимости их реализации.

Python является языком программирования с открытым исходным кодом и поддерживается большим количеством ресурсов, а также имеет документацию высокого качества. Он также может похвастаться большим и активным сообществом разработчиков, готовых предоставить консультации и помощь на всех этапах процесса разработки.

В качестве фреймворка для реализации модели был выбран PyTorch, который разрабатывается группой искусственного интеллекта компании Facebook. Он был выбран из-за ряда преимуществ. PyTorch является продуктом с открытым исходным кодом. PyTorch строит приложения для глубокого обучения на основе динамических графов, с которыми можно работать во время выполнения [7]. Другие популярные фреймворки глубокого обучения работают на статических графах, где вычислительные графы должны быть предварительно построены. Пользователь не имеет возможности увидеть, что происходит при GPU или CPU обработке графа. В то время как в PyTorch есть доступ к каждому уровню вычислений и доступен для просмотра. Дополнительным преимуществом PyTorch является то, что он предоставляет гораздо более глубокое понимание того, что происходит в каждом алгоритме [8]. С таким фреймфорком, имеющим статический вычислительный граф, как TensorFlow, как только декларативно были выражены вычисления, они отправляются на GPU, где они обрабатываются как черный ящик. Но при динамическом подходе есть возможность полностью погрузиться в каждый уровень вычислений и увидеть, что именно происходит на нём.

TensorFlow и PyTorch очень близки по скорости, когда это касается глубокого обучения. Так что потери скорости обучения из-за использование динамических вычислительных графов не происходит [7]. Ключевым моментом является наличие класса, который инкапсулирует все важные выборки данных наряду с выбором архитектуры модели. Это значительно повышает продуктивность и при разработке допускается гораздо меньше ошибок, потому что все, что можно было автоматизировать, было

автоматизировано. Документация к PyTorch также очень хорошая и полезная. Благодаря динамическому вычислительному графу появляется прозрачность для разработчиков и специалистов по данным.

Вычислительный граф в PyTorch определяется во время выполнения, поэтому многие популярные инструменты Python проще использовать в PyTorch [7]. Это огромное преимущество, потому что теперь инструменты отладки из Python, такие как отладчики pdb, ipdb и отладчик PyCharm, могут свободно использоваться для отладки кода PyTorch.

PyTorch имеет одну из важнейших особенностей, известную как декларативный параллелизм данных [8]. Эта функция позволяет использовать «torch.nn.DataParallel» для обёртывания любого модуля. Модуль будет распараллелен по размеру пачек (batch), и эта функция поможет вам легко использовать несколько GPU.

PyTorch предоставляет две высокоуровневые функции: тензорные вычисления (как в NumPy), но с сильным ускорением за счет графических процессоров (GPU) и глубокие нейронные сети, построенные на базе систем автоматической дифференциации [7].

Тензором называют многомерная массив, содержащая элементы одного типа данных. Тензоры PyTorch похожи на массивы NumPy, но также могут обрабатываться на графическом процессоре Nvidia с поддержкой CUDA. PyTorch определяет девять типов CPU тензоров и девять типов GPU тензоров [7]. PyTorch состоит из нескольких основных модулей: модуль автоградиента, модуль оптимизации и модуль нейронных сетей. PyTorch использует метод, называемый автоматической дифференциацией. Рекордер записывает, какие операции были выполнены при прямом проходе сети, а затем повторяет их в обратном порядке для вычисления градиентов. Этот метод особенно эффективен при построении нейронных сетей для экономии времени путем вычисления поправок к параметрам одновременно с прямым проходом по сети. «torch.optim» - модуль, реализующий различные алгоритмы оптимизации, используемые при построении нейронных сетей.

Большинство часто используемых методов уже поддерживается, поэтому нет необходимости реализовывать их с нуля. Модуль автоградиента упрощает определение вычислительных графов и градиентов, но этот модуль может быть слишком низкоуровневым для определения сложных нейронных сетей [7]. В этой задаче может помочь модуль нейронных сетей.

Далее кратко опишем реализацию свёрточной нейронной сети SSD. После чего рассмотрим наиболее интересные и важные участки кода. В качестве набора обучающих данных будет использоваться набор данных Pascal Visual Object Classes (VOC) 2007 и 2012 годов. Это набор данных содержит изображения с двадцатью различными типами объектов, в том числе и объекты типа человек, машина, автобус и мотоцикл, которые необходимо распознавать на изображениях для категоризации и уточнения возгораний согласно поставленной задаче. Из всего набора данных возьмем только изображения, содержащие объекты этих типов. На рисунке 22 отображён фрагмент кода, в котором указываются классы выбираемых из набора данных изображений. Каждое изображение может содержать один или несколько реальных объектов. Описание каждого объект состоит из:

– ограничивающей рамки в абсолютных граничных координатах;

– метки одного из вышеуказанных типов объектов;

– трудность обнаружения объекта (либо 0, что означает «не трудно», либо 1, что означает «трудно»).

```
# Карта меток классов
voc_labels = ('bus', 'car', 'motorbike', 'person')
label_map = {k: v + 1 for v, k in enumerate(voc_labels)}
label_map['background'] = 0
rev_label_map = {v: k for k, v in label_map.items()}

distinct_colors = ['#e6194b', '#3cb44b', '#ffe119', '#0082c8', '#f58231']
label_color_map = {k: distinct_colors[i] for i, k in enumerate(label_map.keys())}
```

Рисунок 22 – Фрагмент кода с картой меток классов

В частности, необходимо загрузить следующие VOC наборы данных: обучающий 2007 года (trainval 2007), обучающий 2012 года (trainval 2012) и проверочный (тестовый) 2007 года (test 2007). Два набора обучающих набора данных будут использоваться для обучения сети, а тестовый будет служить в качестве тестовых данных. После загрузки архивов с данными производим их распаковку, данные из обучающего и тестового набора 2007 года распаковываем в одно и тоже место, то есть объединяем их.

На вход модели будут подаваться три составляющие: изображения, ограничивающие рамки объектов, метки классов объектов. Так как используется вариант SSD300, попадающие в сеть изображения должны быть размером 300 на 300 пикселей и в формате RGB. Также помним, что в качестве базовой сети использовалась VGG-16, предварительно обученная на наборе данных ImageNet, которая уже доступна в модуле torchvison фреймворка PyTorch. Предобработка или преобразование, которое необходимо выполнить для использования модели VGG-16: значения пикселей должны быть в диапазоне от 0 до 1, затем необходимо нормализовать изображение по среднему и стандартному отклонению RGB каналов изображений ImageNet. Средние отклонения: 0.485, 0.456, 0.406. Стандартные отклонения: 0.229, 0.224, 0.225.

Кроме того, PyTorch следует конвенции NCHW, что означает, что количество каналов должно предшествовать размеру изображения. Поэтому изображения, поступающие в модель, должны представлять собой вещественный (float) тензор с размерами (N, 3, 300, 300) и должны быть нормализованы по вышеуказанным средним и стандартным отклонением. N - это размер партии (batch) изображений.

Для каждого изображения необходимо предоставить ограничивающие рамки для присутствующих на нем реальных объектов. Рамки необходимо предоставить в дробных граничных координатах (x_min, y_min, x_max, y_max). Так как количество объектов на любом из изображений может меняться, нельзя использовать тензор фиксированного размера для хранения

ограничивающих рамок для всей партии из N изображений. Следовательно, ограничивающие рамки, должны поступать на вход модели в виде списка длины N, где каждый элемент списка - это вещественный (float) тензор размером (N_o, 4), где N_o - это количество объектов, присутствующих на данном конкретном изображении.

Для каждого изображения необходимо предоставить метки классов для реальных объектов, находящихся на нем. Каждая метка должна быть представлять собой целое число от 1 до 4, представляющее четыре типа объектов, которые необходимо распознавать. Кроме того, будет добавлен фоновый класс с индексом 0, который указывает на отсутствие объекта в ограничивающей рамке. Но, естественно, метка фонового класс на самом деле не будет использоваться ни для одного из объектов набора данных. Опять же, так как количество объектов на любом из изображений может меняться, нельзя использовать тензор фиксированного размера для хранения меток класса для всей партии из N изображений. Поэтому метки классов должны поступать в модель в виде списка длинной N, где каждый элемент списка - это целочисленный (Long) тензор размером N_o, где N_o - это количество объектов, находящихся на данном конкретном изображении.

Как сказано ранее, данные делятся на обучающие и тестовые. Рассмотрим функцию create_data_lists из файла «utils.py». Эта функция является парсером, который собирает данные из папок с файлами и создаёт списки из изображений, ограничивающих рамок и меток объектов на этих изображениях, после чего сохраняет их в JSON файлы. Создаются 3 JSON файла для обучающей и 2 для тестовой выборки. Два файла как для обучающей, так и для тестовой выборки идентичные. Первый содержит список абсолютных путей ко все изображениям разбиения. Во второй файл сохраняется список словарей, содержащих объекты, то есть их ограничивающие рамки в абсолютных граничных координатах, их метки классов и значение трудности обнаружения. i-ый словарь в этом списке содержит объекты, находящиеся на i-ом изображении предыдущего JSON

файла. Дополнительный 3 файл для обучающей выборки содержит карту меток – словарь индекс-метка, в котором хранятся индексы меток и сами метки классов. Этот словарь также доступен в «utils.py» и может быть непосредственно импортирован.

Рассмотрим класс PascalVOCDataset из файла «datasets.py». Этот класс является наследником класса PyTorch Dataset и используется для определения обучающих и тестовых наборов данных. Чтобы его использовать нужно переопределить метод «__len__», который возвращает размер набора данных, и метод «__getitem__», который возвращает очередное изображение, ограничивающие рамки объектов на этом изображении и метки классов для объектов на этом изображении, для этого используются JSON-файлы, которые были получены ранее. Также класс имеет метод «collate_fn», который используется для описания того, как совместить тензоры разных размеров, содержащие изображения, объекты, метки. Для этого используются списки. Также можно заметить, что класс также возвращает трудности обнаружения каждого из этих объектов, но на самом деле они не используются при обучении модели. Они требуются только на этапе оценки для вычисления точности работы обученной модели. Также это даёт возможность полностью отфильтровать сложные для обнаружения объекты из набора данных, чтобы ускорить обучение за счет некоторого снижения точности. Кроме того, внутри этого класса каждое изображение и объекты в нем подлежат некоторым преобразованиям, которые будут рассмотрены далее.

Рассмотрим функцию transform из файла «utils.py», часть которой отображена на рисунке 23. Эта функция применяет различные преобразования ко все изображениям и объектам на них. Случайно корректируются яркость, контраст, насыщенность и оттенок, каждое преобразование применяется с вероятностью в 50% и в случайном порядке. С вероятностью 50% выполняется операция уменьшение масштаба изображения. Это помогает модели обучиться обнаруживать более мелкие

объекты. Уменьшение масштаба изображения должно быть от 1 до 4 раз, то есть такое изображение должно быть от 1 до 4 раз больше оригинального. Окружающее пустое пространство заполняется средним значением данных ImageNet. Также применяется кадрирование изображения, то есть выполнение операции масштабирования. Это помогает модели обучиться обнаружению больших объектов или частей этих объектов. Некоторые объекты могут быть даже полностью вырезаны. Размеры вырезанных частей должны быть в пределах от 0.3 до 1 от исходных размеров. Соотношение их сторон должны иметь значения между 0.5 и 2. Каждая часть изображения вырезается таким образом, чтобы оставалась, по крайней мере, одна ограничивающая рамка объекта, которая с этой вырезанной частью имеет один из следующих коэффициентов наложения Жаккара: 0, 0.1, 0.3, 0.5, 0.7, 0.9, выбираемый случайным образом. Кроме того, все оставшиеся ограничивающие рамки, центры которых больше не находятся на вырезанной части изображения, отбрасываются. Также существует вероятность того, что к изображению вообще не будет применяться данная операция.

```
if split == 'TRAIN':
    new_image = photometric_distort(new_image)

    new_image = FT.to_tensor(new_image)

    if random.random() < 0.5:
        new_image, new_boxes = expand(new_image, boxes, filler=mean)

    new_image, new_boxes, new_labels, new_difficulties = random_crop(new_image, new_boxes, new_labels,
                                                                     new_difficulties)

    new_image = FT.to_pil_image(new_image)

    if random.random() < 0.5:
        new_image, new_boxes = flip(new_image, new_boxes)

new_image, new_boxes = resize(new_image, new_boxes, dims=(300, 300))

new_image = FT.to_tensor(new_image)

new_image = FT.normalize(new_image, mean=mean, std=std)

return new_image, new_boxes, new_labels, new_difficulties
```

Рисунок 23 – Фрагмент кода функции transform

С вероятностью 50% применяется горизонтальный переворот изображения. Изменяется размер изображения до 300 на 300 пикселей. Все рамки объектов конвертируются из абсолютных в дробные граничные координаты. На всех этапах модели все граничные и центрально-размерные координаты используются в дробной форме. Изображение нормализуется по среднему и стандартному отклонениям данных ImageNet, которые использовалось для предварительного обучения базовых слоёв из модели VGG-16. Эти преобразования играют решающую роль в получении точных результатов, так как вносят различные помехи и дополнительные варианты расположения объектов на изображениях, что улучшает точность итоговой обученной сети.

Описанный выше класс для набора данных, PascalVOCDataset, будет использоваться PyTorch DataLoader, который находится в файле «train.py» для создания и подачи партий данных в модель для обучения или оценки. Так как количество объектов на разных изображениях разное, их ограничивающие рамки, метки и показатели трудности не могут быть просто сложены вместе в партию изображений. Нет возможности понять, какие объекты к какому изображению принадлежат. Вместо этого просто передаем аргументу «collate_fn» загрузчика данных функцию упорядочивания, созданную ранее как метод класса PascalVOCDataset, которая инструктирует загрузчик о том, как он должен комбинировать эти тензоры различного размера. Самым простым вариантом будет использование списков Python.

Рассмотрим класс VGGBase из файла «model.py», конструктор которого представлен на рисунке 24. Здесь создаются базовые свёрточные слои нейронной сети. Класс наследуется от nn.Module, так как в PyTorch нейронные сети и слои должны наследоваться от этого класса, в котором представлена базовая структура для них. В конструкторе класса определяются свёрточные слои и задаются их параметры. Всего создаётся 5 групп слоёв и 6 с 7 слои. Создание слоёв производится в соответствии с рассмотренной архитектурой. Также в классе для модели задаётся метод

прямого распространения, в котором используется в качестве функции активации ReLU из набора функций модуля nn. Слои инициализируются параметрами из предварительно обученной нейронной сети VGG-16 с помощью метода load_pretrained_layers. Как было сказано ранее, эти параметры доступны в PyTorch в модуле torchvision. Откуда эти параметры копируются в сеть. Это просто для групп слоёв с 1 по 5. Однако оригинальная версия VGG-16 не содержит 6 и 7 свёрточных слоёв. Поэтому конвертируем исходные слои в свёрточные, и производим децимацию полученных слоёв для уменьшения их размеров с помощью функции decimate из файла «utils.py». Конвертация и децимация выполняются по алгоритмам, описанным в втором разделе. Особенно интересны в данном случае карты низкоуровневые карты признаков, полученные в результате прохода через свёрточные слои 4_3 и 7. Они будут использования на последующих этапах. Поэтому метод прямого распространения по базовым слоям возвращает эти карты признаков.

```
def __init__(self):
    super(VGGBase, self).__init__()

    self.conv1_1 = nn.Conv2d(3, 64, kernel_size=3, padding=1)  # stride = 1
    self.conv1_2 = nn.Conv2d(64, 64, kernel_size=3, padding=1)
    self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)

    self.conv2_1 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
    self.conv2_2 = nn.Conv2d(128, 128, kernel_size=3, padding=1)
    self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)

    self.conv3_1 = nn.Conv2d(128, 256, kernel_size=3, padding=1)
    self.conv3_2 = nn.Conv2d(256, 256, kernel_size=3, padding=1)
    self.conv3_3 = nn.Conv2d(256, 256, kernel_size=3, padding=1)
    self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2, ceil_mode=True)  # O

    self.conv4_1 = nn.Conv2d(256, 512, kernel_size=3, padding=1)
    self.conv4_2 = nn.Conv2d(512, 512, kernel_size=3, padding=1)
    self.conv4_3 = nn.Conv2d(512, 512, kernel_size=3, padding=1)
    self.pool4 = nn.MaxPool2d(kernel_size=2, stride=2)

    self.conv5_1 = nn.Conv2d(512, 512, kernel_size=3, padding=1)
    self.conv5_2 = nn.Conv2d(512, 512, kernel_size=3, padding=1)
    self.conv5_3 = nn.Conv2d(512, 512, kernel_size=3, padding=1)
    self.pool5 = nn.MaxPool2d(kernel_size=3, stride=1, padding=1)

    # Конвертированные в сверточные слои
    self.conv6 = nn.Conv2d(512, 1024, kernel_size=3, padding=6, dilation=6)

    self.conv7 = nn.Conv2d(1024, 1024, kernel_size=1)
```

Рисунок 24 – Конструктор класса базовых слоёв

Перейдем к рассмотрению класса AuxiliaryConvolutions из того же файла «model.py», фрагмент которого представлен на рисунке 25. В нём реализуются дополнительные свёрточные слои для получения высокоуровневых карт признаков. В конструкторе класса объявляются свёрточные слои и задаются их параметры согласно архитектуре модели. Также задаётся метод «init_conv2d», который использует равномерную инициализацию Ксавьера (uniform Xavier initialization), реализованную в модуле nn.init, для задания начальных значений параметров этих слоев [9]. Также инициализируются и смещения, начальное значение которых задается равным нулю. Задаем метод для прямого распространения. В этих части модели особенно интересны высокоуровневые карты признаков, получаемые при прохождении данными свёрточных слоёв 8_2, 9_2, 10_2 и 11_2, которые возвращает метод прямого распространения для использования на последующих этапах.

```
def __init__(self):
    super(AuxiliaryConvolutions, self).__init__()

    self.conv8_1 = nn.Conv2d(1024, 256, kernel_size=1, padding=0)
    self.conv8_2 = nn.Conv2d(256, 512, kernel_size=3, stride=2, padding=1)

    self.conv9_1 = nn.Conv2d(512, 128, kernel_size=1, padding=0)
    self.conv9_2 = nn.Conv2d(128, 256, kernel_size=3, stride=2, padding=1)

    self.conv10_1 = nn.Conv2d(256, 128, kernel_size=1, padding=0)
    self.conv10_2 = nn.Conv2d(128, 256, kernel_size=3, padding=0)

    self.conv11_1 = nn.Conv2d(256, 128, kernel_size=1, padding=0)
    self.conv11_2 = nn.Conv2d(128, 256, kernel_size=3, padding=0)

    # Инициализация параметров
    self.init_conv2d()
```

Рисунок 25 – Фрагмент кода, создающего дополнительные слои

Рассмотрим класс PredictionConvolutions из файла «model.py», который реализует финальные свёрточные слои для предсказаний, фрагмент кода

класса представлен на рисунке 26. Здесь создаются свёртки для локализации и предсказания классов, применяемые к картам признаков из свёрточных слоёв 4_3, 7, 8_2, 9_2, 10_2 и 11_2. В конструкторе класса заданы слои согласно архитектуре, на каждую карту признаков по 2 слоя. Также создан словарь, хранящий количество приоров на ячейку карты признаков для каждого слоя. Слои здесь инициализируются таким же образом, как и дополнительные. В методе прямого распространения во время прохода по слоям полученные выходные свёртки преобразуются описанным в разделе проектирования способом в массивы из строк, каждая из которых содержит данные по одному предсказания (набор отклонений от приора или набор оценок классов). В конце все массивы объединяются в общие 2 стека (предсказания для локализации и для классификации). Стоит обратить внимание, что изменение формы тензоров в PyTorch возможно только в том случае, если исходный тензор хранится в смежных блоках памяти, для этого используется функция contiguous. Метод прямого распространения возвращает итоговые два стека с предсказаниями. Как и ожидалось, в нашем случае стеки будут иметь размеры 8732, 4 и 8732, 5 соответственно.

```
self.n_classes = n_classes

n_boxes = {'conv4_3': 4,
           'conv7': 6,
           'conv8_2': 6,
           'conv9_2': 6,
           'conv10_2': 4,
           'conv11_2': 4}

# Слои для локализации
self.loc_conv4_3 = nn.Conv2d(512, n_boxes['conv4_3'] * 4, kernel_size=3, padding=1)
self.loc_conv7 = nn.Conv2d(1024, n_boxes['conv7'] * 4, kernel_size=3, padding=1)
self.loc_conv8_2 = nn.Conv2d(512, n_boxes['conv8_2'] * 4, kernel_size=3, padding=1)
self.loc_conv9_2 = nn.Conv2d(256, n_boxes['conv9_2'] * 4, kernel_size=3, padding=1)
self.loc_conv10_2 = nn.Conv2d(256, n_boxes['conv10_2'] * 4, kernel_size=3, padding=1)
self.loc_conv11_2 = nn.Conv2d(256, n_boxes['conv11_2'] * 4, kernel_size=3, padding=1)

# Слои для классификации
self.cl_conv4_3 = nn.Conv2d(512, n_boxes['conv4_3'] * n_classes, kernel_size=3, padding=1)
self.cl_conv7 = nn.Conv2d(1024, n_boxes['conv7'] * n_classes, kernel_size=3, padding=1)
self.cl_conv8_2 = nn.Conv2d(512, n_boxes['conv8_2'] * n_classes, kernel_size=3, padding=1)
self.cl_conv9_2 = nn.Conv2d(256, n_boxes['conv9_2'] * n_classes, kernel_size=3, padding=1)
self.cl_conv10_2 = nn.Conv2d(256, n_boxes['conv10_2'] * n_classes, kernel_size=3, padding=1)
self.cl_conv11_2 = nn.Conv2d(256, n_boxes['conv11_2'] * n_classes, kernel_size=3, padding=1)
```

Рисунок 26 – Фрагмент кода конструктора класса слоёв для предсказаний

Рассмотрим класс SSD300 из файла «model.py», фрагмент кода которого представлен на рисунке 27. В этом классе объединяются базовые, дополнительные и предсказательные слои для того чтобы получить полную модель нейронной сети SSD. В конструкторе класса создаем объекты остальных трёх классов. Здесь есть небольшая деталь - признаки самого низкого уровня, то есть из слоя 4_3, ожидаются в значительно отличающемся от вышестоящих аналогов числовом масштабе. Поэтому авторы SSD рекомендуют использовать L2-нормализацию, а затем перемасштабирование каждого из каналов по некоторому коэффициенту. Коэффициент масштабирования изначально устанавливается 20, но изучается для каждого канала во время обратного распространения. Метод прямого распространения возвращает стеки с предсказаниями локализации и классов.

```
conv4_3_feats, conv7_feats = self.base(image)  # (N, 512, 38, 38), (N, 1024, 19, 19)

# Изменение масштаба карты признаков 4_3 после L2 нормировки
norm = conv4_3_feats.pow(2).sum(dim=1, keepdim=True).sqrt()  # (N, 1, 38, 38)
conv4_3_feats = conv4_3_feats / norm  # (N, 512, 38, 38)
conv4_3_feats = conv4_3_feats * self.rescale_factors  # (N, 512, 38, 38)

conv8_2_feats, conv9_2_feats, conv10_2_feats, conv11_2_feats = \
    self.aux_convs(conv7_feats)  # (N, 512, 10, 10),  (N, 256, 5, 5), (N, 256, 3, 3), (N, 256, 1, 1)

locs, classes_scores = self.pred_convs(conv4_3_feats, conv7_feats, conv8_2_feats, conv9_2_feats, conv10_2_feats,
                                       conv11_2_feats)  # (N, 8732, 4), (N, 8732, n_classes)

return locs, classes_scores
```

Рисунок 27 – Фрагмент кода класса SSD300

На рисунке 28 представлена диаграмма классов, отображающая основные классы и взаимосвязи между ними в реализованной модели свёрточной нейронной сети. Для создания приоров была разработана функция create_prior_boxes из файла «model.py». Эта функция создает приоры в центрально-размерных координатах для карт признаков из слоёв 4_3, 7, 8_2, 9_2, 10_2 и 11_2, в этом порядке. Для каждой ячейки карты признаков создается установленное количество приоров, не забывая о дополнительном приоре с масштабом равным геометрическому среднему масштабов приоров предыдущей карты признаков и текущей. Порядок 8732 полученных таким образом приоров очень важен, так как он должен соответствовать порядку сложенных в стеки предсказаний для этих приоров.

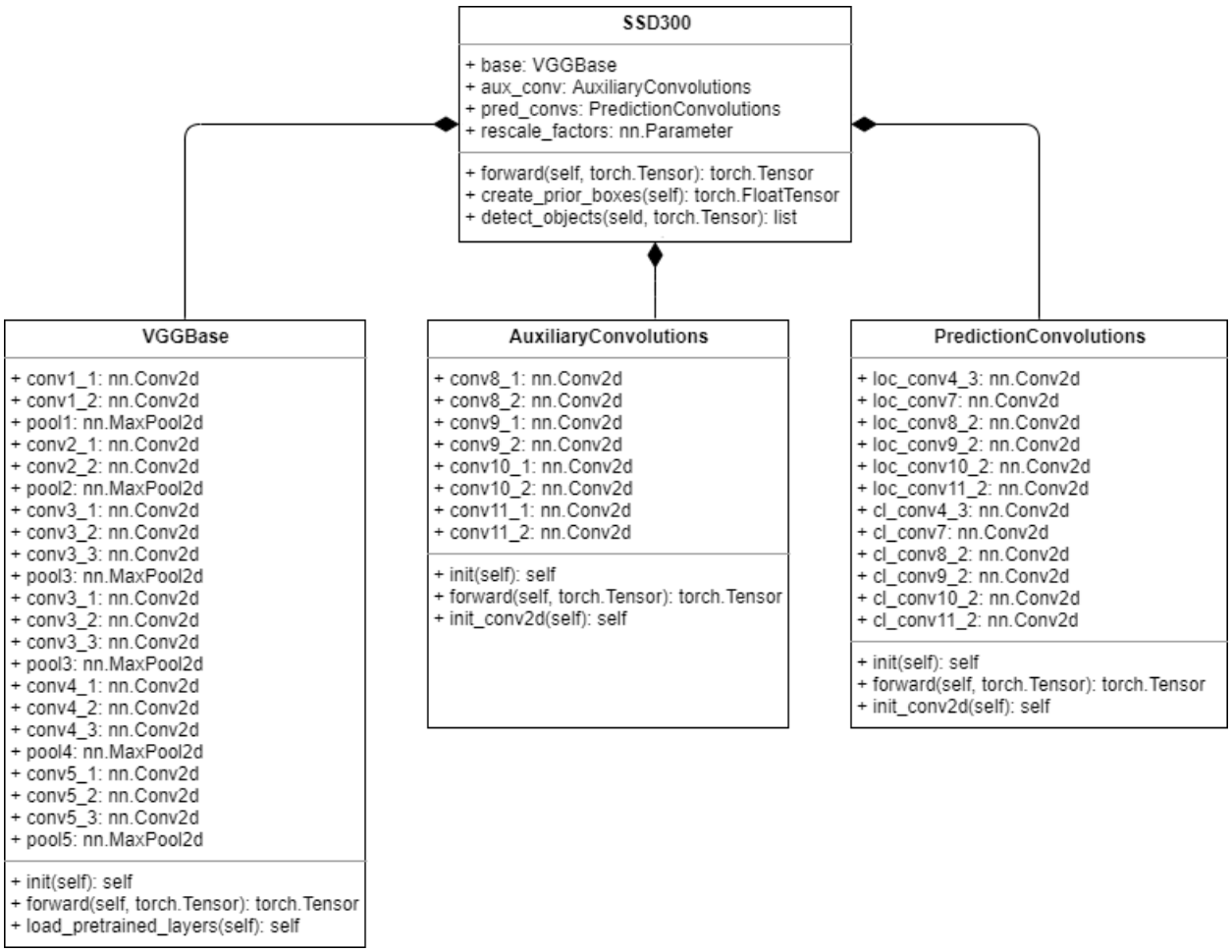

Рисунок 28 – Диаграмма классов

Рассмотрим класс MultiBoxLoss из файла «model.py», который используется для вычисления общих потерь локализации и классификации. Два пустых тензора создаются для хранения целевых координат реальных объектов и их меток классов, то есть целевые показатели для 8732 предсказанных рамок на каждом изображении. Мы находим реальный объект с максимальным коэффициентом Жаккара для каждого приора, и сохраняем их в object_for_each_prior. Может случиться редкая ситуация, когда не для всех реальных объектов получилось найти совпадение. Для того чтобы избежать её также находим приор с максимальным наложением для каждого объекта и храним их в тензоре prior_for_each_object. После чего явно добавляем эти совпадения в object_for_each_prior и искусственно устанавливаем их коэффициент наложения на значение выше порога, чтобы они не удалились. Основываясь на совпадениях в object_for_each_prior,

устанавливаем соответствующие метки, то есть целевые метки для предсказания класса, для каждого из 8732 приоров. Для тех приоров, которые существенно не пересекаются со своими совпадающими объектами, в качестве метки устанавливается фоновый класс. Также, мы преобразуем координаты 8732 совпавших объектов в тензоре object_for_each_prior к виду набора смещений $(g_{cx}, g_{cy}, g_w, g_h)$ по отношению к этим приорам, чтобы сформировать цели для локализации. Не все из этих 8732 целей локализации значимые. Как было сказано в разделе проектирования, только к предсказаниям, вытекающим из приоров не фонового класса, будет применена операция регрессии. Потери локализации - это значения функции Smooth L1 по положительным совпадениям.

Далее применяем метод Hard Negative Mining, выстраиваем в ряд предсказания, сопоставленные с фоновым классом, то есть сортируем отрицательные совпадения по индивидуальным потерям перекрестной энтропии. Потери классификации - это потеря перекрестной энтропии над положительными совпадениями и самыми отрицательными совпадениями. Тем не менее, эти потери усредняются только по количеству положительных совпадений. Multibox потери - это сумма двух потерь (локализации и классификации), объединенных в соотношении α. В нашем случае они просто складываются потому, что α = 1.

Таким образом, в этом подразделе были рассмотрены основные классы и функции реализации спроектированной модели свёрточной нейронной сети SSD. Были проанализированы выбранные для реализации язык программирования Python и фреймворк PyTorch. Выявлены их преимущества для реализации спроектированной архитектуры и рассмотрены инструменты, которые они предоставляют. В следующем подразделе перейдем к обучению модели реализованной программой, её тестированию и оценки точности.

3.2 Тестирование свёрточной нейронной сети для распознавания и локализации объектов на изображениях

Прежде чем начать обучение модели, обязательно необходимо распаковать все файлы данных для обучения и оценки. Для этого предназначен скрипт create_data_lists.py, который вызывает рассмотренную ранее функцию для создания JSON-файлов, указывая ей пути местонахождения папок VOC2007 и VOC2012, в которых хранятся распакованные данные.

Обучение осуществляется разработанным скриптом «train.py». Параметры модели (и ее обучения) находятся в начале скрипта, так что их можно легко проверить или изменить их при необходимости, выбранные параметры отображены на рисунке 29. Помимо стандартных параметров вроде количества итераций и размера партий изображений есть еще ряд дополнительных. В параметр «device» устанавливается устройство, на котором будет происходить обучение, если доступен графический процесс, то будет выбран он, иначе центральный. Также есть параметр для указания чекпоинта модели, если необходимо продолжить обучение с некоторой эпохи. Количество работников (workers) указывает количество потоков загрузки данных для загрузчика данных. В скрипте содержится функция «train», которая осуществляет одну эпоху обучения.

```
n_classes = len(label_map)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Параметры обучения
checkpoint = None
batch_size = 8  # размер партии
iterations = 80000  # количество итераций
workers = 4  # количество потоков загрузки ищзображений для DataLoader
print_freq = 200  # частота вывода информации при обучении
lr = 1e-3  # скорость обучения
decay_lr_at = [53000, 66000]  # номера итераций на которых уменьшается скорость обучения
decay_lr_to = 0.1  # на сколько уменьшать скорость обучения
momentum = 0.9  # импульс
weight_decay = 5e-4  # сокращение веса
grad_clip = None
```

Рисунок 29 – Фрагмент кода с параметрами обучения

Чтобы обучить модель с нуля, запускаем скрипт «train.py» с помощью интерпретатора Python. Вывод скрипта в консоль в процессе обучения представлен на рисунке 30. В начале строки вывода указывается номер эпохи и количество обработанных партий изображений. Затем указывается время обработки партии изображений, то есть их прямого и обратного распространения, в скобка при этом указывается среднее время обработки. Затем выводится время загрузки данных: для текущей партии и среднее. После чего выводятся рассчитанные потери: текущие и средние.

```
Epoch: [0][0/1012]      Batch Time 5.174 (5.174)       Data Time 1.607 (1.607) Loss 18.7672 (18.7672
Epoch: [0][200/1012]    Batch Time 0.210 (0.342)       Data Time 0.000 (0.106) Loss 5.0084 (7.1515)
Epoch: [0][400/1012]    Batch Time 0.206 (0.327)       Data Time 0.000 (0.099) Loss 4.7321 (5.8275)
Epoch: [0][600/1012]    Batch Time 0.205 (0.321)       Data Time 0.000 (0.096) Loss 4.6712 (5.2959)
Epoch: [0][800/1012]    Batch Time 0.733 (0.317)       Data Time 0.525 (0.094) Loss 3.4891 (4.9640)
Epoch: [0][1000/1012]   Batch Time 0.207 (0.315)       Data Time 0.000 (0.092) Loss 3.6581 (4.7532)
Epoch: [1][0/1012]      Batch Time 1.406 (1.406)       Data Time 1.192 (1.192) Loss 3.7011 (3.7011)
Epoch: [1][200/1012]    Batch Time 0.202 (0.322)       Data Time 0.000 (0.101) Loss 3.7534 (3.8127)
Epoch: [1][400/1012]    Batch Time 0.240 (0.316)       Data Time 0.017 (0.096) Loss 3.3758 (3.7441)
Epoch: [1][600/1012]    Batch Time 0.228 (0.312)       Data Time 0.000 (0.091) Loss 3.0085 (3.6796)
Epoch: [1][800/1012]    Batch Time 0.239 (0.312)       Data Time 0.000 (0.092) Loss 3.5823 (3.6457)
Epoch: [1][1000/1012]   Batch Time 0.785 (0.312)       Data Time 0.569 (0.092) Loss 4.1332 (3.6063)
Epoch: [2][0/1012]      Batch Time 1.947 (1.947)       Data Time 1.721 (1.721) Loss 3.7012 (3.7012)
Epoch: [2][200/1012]    Batch Time 0.220 (0.327)       Data Time 0.000 (0.105) Loss 3.4407 (3.4935)
Epoch: [2][400/1012]    Batch Time 0.220 (0.315)       Data Time 0.000 (0.094) Loss 3.3287 (3.4274)
Epoch: [2][600/1012]    Batch Time 0.472 (0.316)       Data Time 0.251 (0.096) Loss 3.0280 (3.4073)
Epoch: [2][800/1012]    Batch Time 0.225 (0.318)       Data Time 0.013 (0.098) Loss 2.9071 (3.3759)
Epoch: [2][1000/1012]   Batch Time 0.210 (0.320)       Data Time 0.000 (0.100) Loss 3.4384 (3.3524)
```

Рисунок 30 – Вывод информации при обучении модели

Разработчики модели нейронной сети SSD рекомендуют использовать стохастический градиентный спуск партиями по 32 изображения, с начальной скоростью обучения 1e-3, импульсом 0,9 и сокращением веса 5e-4. После нескольких экспериментов с обучением, выяснилось, что в условиях задачи использование партий размером 8 изображений повышает стабильность, так как при таком размере партий не происходит взрыва градиентов (exploding gradients). Взрыв градиентов - это проблема, при которой накапливаются большие градиенты ошибок, что приводит к очень большому обновлению весов в модели нейронной сети во время обучения. Это приводит к тому, что модель становится нестабильной и не может учиться на тренировочных данных [10]. Эту проблему можно решить с помощью уменьшения размера партий либо установлением ограничения на

размер градиента во время обучения [10]. Для ограничения в скрипте для обучения есть параметр «grad_clip».

Авторы также удвоили скорость обучения параметров смещения. Как видно из кода, это легко сделать в PyTorch, передав отдельные группы параметров аргументу «params» оптимизатора SGD. Для этого проходим по параметрам модели, выделяем смещения и при передаче в оптимизатор увеличиваем для них скорость обучения.

Для данной модели и набора данных рекомендуется проводить обучение в объеме 80000 итераций при начальной скорости обучения. Затем скорость уменьшается на 90% для дополнительных 20000 итераций, дважды. И всего получается 120000 итераций. Но это для 20 типов объектов оригинального набора данных, так как используется только 4 типа для решения поставленной задачи, то было принято решения установить количество итераций 80000, уменьшение скорости обучения происходит после 53000 и 66000 итераций.

Обучение производились на облачном сервисе от компании Google под названием Colab. Сервис предоставляет виртуальную машину с достаточными для обучения нейронной сети вычислительными ресурсами: 12 гигабайт оперативной памяти, одноядерный двухпоточный процессор Xeon 2.3 гигагерц, графический процессор Nvidia Tesla K80 с 2496 ядрами CUDA и 12 гигабайтами GDDR5 памяти. Обучение естественно проводилось на графическом процессоре, так как это намного ускоряет процесс обучения. Обучение продолжалось на протяжении 79 эпох и заняло около 9 часов.

Теперь необходимо провести оценку обученной модели нейронной сети. Для это используем скрипт «eval.py», реализованный для оценки средней точности модели по каждому из распознаваемых объектов. Параметры загрузки данных и чекпоинта для оценки модели находятся в начале скрипта, поэтому их можно легко проверить или изменить при желании. Чтобы начать оценку, необходимо запустите функцию evaluation с указанием загрузчика данных и чекпоинта модели. Предсказания для

каждого изображения в тестовом наборе получаются и парсятся методом detect_objects из чекпоинта, который реализует процесс их получения, описанный в разделе проектирования. Оценка должна быть выполнена при параметрах min_score 0.01, max_overlap 0.45 и top_k 200, чтобы обеспечить справедливое сравнение результатов работы модели с другими реализациями. Полученные предсказания оцениваются по отношению к реальным объектам. Метрикой оценки является Mean Average Precision (mAP). Для вычисления этой метрики используется функция calculate_mAP из файла «utils.py». Как и положено, при расчете mAP будут игнорироваться сложные для распознавания объекты. Но, тем не менее, важно включать их из набора данных для оценки, так как если модель действительно обнаружит объект, который считается сложным, то его нельзя считать ложным срабатыванием.

После выполнения оценки получены результаты, представленные на рисунке 31. Представлены средние точности работы модели по каждому типу объектов, а также общая средняя точность модели 0.829.

```
Evaluating: 100% 43/43 [14:14<00:00, 19.88s/it]
{'bus': 0.8556174039840698,
 'car': 0.8602738380432129,
 'motorbike': 0.8058657646179199,
 'person': 0.7924215197563171}

Mean Average Precision (mAP): 0.829
```

Рисунок 31 – Результаты оценки точности работы модели

Проведём тестирование обученной модели на различных изображениях с выводом рамок. Для этого воспользуемся разработанным скриптом «detect.py», фрагмент кода которого представлен на рисунке 32. В начале скрипта необходимо указать на модель (сохранённый чекпоинт), которая будет использоваться для предсказания в специальном параметре. Затем можно использовать функцию «detect» для распознавания и визуализации объектов на RGB-изображении. Эта функция сначала предобрабатывает изображение, изменяя его размер и нормализуя его каналы RGB в

соответствии с требованиями модели. Затем она получает необработанные предсказания от модели, которые парсятся методом detect_objects из класса модели. Результаты преобразуются из дробных в абсолютные граничные координаты, их метки декодируются с помощью карты меток и визуализируются на изображении.

```
# Рамки
box_location = det_boxes[i].tolist()
draw.rectangle(xy=box_location, outline=label_color_map[det_labels[i]])
draw.rectangle(xy=[l + 1. for l in box_location], outline=label_color_map[
    det_labels[i]])

# Текст
textr = det_labels[i].upper() + ' ' + str(round(det_scores[i],2))
text_size = font.getsize(textr)
text_location = [box_location[0] + 2., box_location[1] - text_size[1]]
textbox_location = [box_location[0], box_location[1] - text_size[1], box_location[0] + text_size[0] + 4.,
                    box_location[1]]
draw.rectangle(xy=textbox_location, fill=label_color_map[det_labels[i]])
draw.text(xy=text_location, text=textr, fill='white',
          font=font)
```

Рисунок 32 – Фрагмент кода функции detect

Результаты тестирования на различных изображениях с различными типами объектов представлены на рисунках 33, 34, 35 и 36. После отображения по координатам ограничивающей рамки объекта, устанавливается шрифт для текста. Кроме метки класса выводится также и очки от 0 до 1, выражающие то насколько модель уверенна в отношении класса данного объекта.

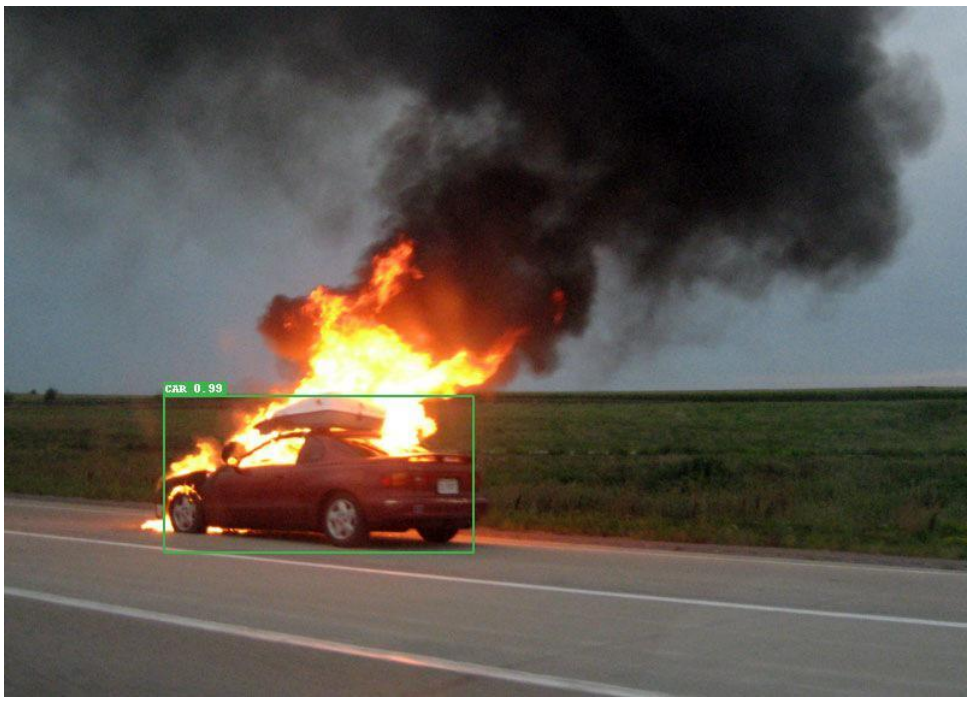

Рисунок 33 – Результат применения обученной модели (1)

Рисунок 34 – Результат применения обученной модели (2)

Рисунок 35 – Результат применения обученной модели (3)

Рисунок 36 – Результат применения обученной модели (4)

После применения модели к различным изображениям с различными объектами, можно сказать, что она справляется с поставленной задачи, а

именно распознаёт и локализует объекты установленных классов, тем самым предоставляет уточняющую информацию о возгораниях.

По итогам этого подраздела было произведено обучение модели сверхточной нейронной сети. Полученная модель была оценена: с помощью метрики была вычислена средняя точность работы модели на различных типах объектов. После чего было проведено тестирование, а именно применение обученной модели к различным изображениям и рассмотрение полученных результатов.

В данном разделе была рассмотрена реализация свёрточной нейронной сети для локализации и распознавания объектов на изображениях, а также средства её реализации. Рассмотрен процесс обучения нейронной сети и проведения её оценки. После чего полученная нейронная сеть была протестирована на изображениях, содержащих различные объекты. Реализованная нейронная сеть выполняет все поставленные задачи.

Заключение

Выполненная бакалаврская работа посвящена реализации алгоритма идентификации объектов на изображении для уточнения информации о возгораниях. В качестве алгоритма была выбрана свёрточная нейронная сеть, которая должна обнаруживать объекты на изображениях, на которых запечатлены возгорания.

Для того чтобы достигнуть цели был выполнен ряд задач. Был проведен анализ технологии искусственных нейронных сетей, а затем свёрточных сетей. Всё это было сделано в рамках анализа применения СНС для локализации и распознавания объектов на изображениях. Также была разобрана базовая архитектура СНС.

Был произведен поиск подходящей для реализации модели СНС. И в качестве модели была выбрана Single Shot Multibox Detector (SSD), которая предназначена для обнаружения объектов на изображениях и обладает высокими показателями точности. После разбора архитектуры данной модели и всех применяемых в ней методах был осуществлен переход к реализации.

Был обоснован выбранный для реализации язык программирования Python и фреймворк PyTorch, их особенности, преимущества и предоставляемые инструменты. Была реализована и обучена свёрточная нейронная сеть. На вход данной сети подаются изображения: после процесса обнаружения объектов на выходе получаются исходные изображения с обнаруженными объектами, и оценка, выражающая степень уверенности сети. Полученная модель нейронной сети была оценена по точности обнаружения каждого класса объектов и была вычислена общая средняя точность. Было проведено тестирование сети на различных изображениях и показана её работоспособность. Данная работа продемонстрировала неограниченную возможность применения свёрточных нейронных сетей в

области пожарной безопасности для автоматического уточнения информации о возгораниях.

Список используемых источников

1. Саймон Х. Нейронные сети: полный курс / Саймон Хайкин – Издательский дом Вильямс, 2006. – 1104 с.
2. An Intuitive Explanation of Convolutional Neural Networks, 2016 // Ujjwalkarn [Электронный ресурс]: сайт о нейронных сетях. URL: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/ (дата обращения 15.03.2020).
3. Banks D. Classification, Clustering, and Data Mining Applications: Proceedings of the Meeting of the International Federation of Classification / David Banks, Phipps Arabie - Springer-Verlag Berlin Heidelberg. – 2004.-Jul. -C 658.
4. Chen L. Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs / Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, Alan L. Yuille – Springer International Publishing. – 2015.-Jun. -C 14.
5. Hoiem D. Diagnosing error in object detectors / Derek Hoiem, Yodsawalai Chodpathumwan, Qieyun Dai – Springer International Publishing. – 2012.-Feb. -C 23.
6. Jason Brownlee. A Gentle Introduction to Exploding Gradients in Neural Networks, 2019 // Machine Learning Mastery [Электронный ресурс]: сайт с научными статьями. URL: https://machinelearningmastery.com/exploding-gradients-in-neural-networks/ (дата обращения 05.04.2020).
7. Jayaraman B. Artificial Neural Networks - Theory and Applications / Jayaraman Balaji – Springer International Publishing. – 2017.-Sep. -C 54.
8. Jordi Torres. Learning process of a neural network, 2018 // Medium [Электронный ресурс]: сайт с научными статьями. URL: https://towardsdatascience.com/how-do-artificial-neural-networks-learn-773e46399fc7 (дата обращения 15.03.2020).

9. Joshua H. Aircraft System Identification Using Artificial Neural Networks With Flight Test Data / Joshua Harris – Springer International Publishing. – 2016.-Jun. -C. 150.

10. Kaiming H. Deep residual learning for image recognition / Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun – IEEE. – 2016.-Jun. -C 78.

11. Kenton K. Aircraft System Identification Using Artificial Neural Networks / Kenton Kirkpatrick, Jim May Jr, John Valasekz – Springer International Publishing. – 2013.-Jan. -C 124.

12. Learning pytorch with examples // PyTorch [Электронный ресурс]: официальный сайт фреймворка PyTorch. URL: https://pytorch.org/tutorials/beginner/pytorch_with_examples.html (дата обращения 25.03.2020).

13. Li W. Automatic Localization and Count of Agricultural Crop Pests Based on an Improved Deep Learning Pipeline / Li Wen, Chen Pen – Springer International Publishing. – 2019.-Sep. -C 114.

14. Liu W. SSD: Single Shot MultiBox Detector / Liu Wei, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy – Springer International Publishing. – 2016.-May. -C 90.

15. Mayank Jain. The Basics of Neural Networks, 2019 // Medium [Электронный ресурс]: сайт с научными статьями. URL: https://medium.com/datadriveninvestor/the-basics-of-neural-networks-304364b712dc (дата обращения 10.03.2020).

16. Naoto Hieda. How neural networks are trained, 2018 // GitHub [Электронный ресурс]: веб-сервис для хостинга IT-проектов. URL: https://ml4a.github.io/ml4a/how_neural_networks_are_trained/ (дата обращения 12.03.2020).

17. Neural Network Programming - Deep Learning with PyTorch // DeepLizard [Электронный ресурс]: сайт с курсами по PyTorch. URL: https://deeplizard.com/learn/video/iTKbyFh-7GM (дата обращения 27.03.2020).

18. Prateek Joshi. Understanding Xavier Initialization In Deep Neural Networks, 2016 // Perpetual Enigma [Электронный ресурс]: авторский сайт ученного. URL: https://prateekvjoshi.com/2016/03/29/understanding-xavier-initialization-in-deep-neural-networks/ (дата обращения 01.04.2020).

19. Redmon J. You only look once: unified, real-time object detection / Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi – IEEE. – 2016.-Oct. -C 23.

20. Ren S. Faster R-CNN: towards real-time object detection with region proposal networks / Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun – IEEE. – 2017.-Jan. -C 132.

21. Sean B. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) / Sean Bell, C. Lawrence Zitnick, Kavita Bala, Ross Girshick – Springer International Publishing. – 2016.-Jul. -C 109.

22. Sermanet P. Overfeat: integrated recognition, localization and detection using convolutional networks / Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, Yann LeCun – Springer International Publishing. – 2014.-Feb. -C 16.

23. Shiva Verma. Understanding 1D and 3D Convolution Neural Network, 2019 // Towards data science [Электронный ресурс]: архив статей по науке о данных. URL: https://towardsdatascience.com/understanding-1d-and-3d-convolution-neural-network-keras-9d8f76e29610 (дата обращения 20.03.2020).

24. Simonyan K. Very deep convolutional networks for large-scale image recognition / Karen Simonyan, Andrew Zisserman – Springer International Publishing. – 2015.-Apr. -C 14.

25. Uijling J. Selective Search for Object Recognition / Uijlings J.R.R., van de Sande K.E.A – Springer International Publishing. – 2013.-Apr. -C 154.

Приложение А

Исходный код реализованной модели нейронной сети

Исходный код реализованной модели свёрточной нейронной сети находится на компакт-диске.