МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

РАБОТА ПРОВЕРЕНА

Рецензент
Доцент кафедры ВМиИТ
ФГБОУ ВО «ЧелГУ»,
к.ф.-м.н.

_____ А.Ю.Маковецкий

«___»_____ 2024 г.

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Разработка нейронной сети для классификации задач ОГЭ по способам решения

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.04.02.2024.308-1400.ВКР

Научный руководитель,
доцент кафедры СП, к.ф.-м.н.
_____ Т.Ю. Маковецкая

Автор работы,
студент группы КЭ-220
_____ И.М. Сысоева

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы магистранта

студентке группы КЭ-220
Сысоевой Ирине Михайловне,
обучающейся по направлению
02.04.02 «Фундаментальная информатика и информационные технологии»
(магистерская программа «Машинное обучение и анализ больших данных»)

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)
Разработка нейронной сети для классификации задач ОГЭ по способам решения.
2. **Срок сдачи студентом законченной работы:** 20.05.2024 г.
3. **Исходные данные к работе**

3.1. Гудфеллоу Я., Бенжио И., Курвиль А. Глубокое обучение. – Москва: ДМК Пресс, 2018. – 652 с.
3.2. Антонио Д., Суджит П. Библиотека Keras – инструмент глубокого обучения. Реализация нейронных сетей с помощью библиотек Theano и TensorFlow. – Москва: ДМК Пресс, 2018. – 294 с.
3.3. Селянкин В.В. Компьютерное зрение. Анализ и обработка изображений: учебник для вузов. – Санкт-Петербург: Лань, 2021. – 152с.
3.4. Федеральная служба по надзору в сфере образования и науки. ФГБНУ «Федеральный институт педагогических измерений». [Электронный ресурс] URL: https://fipi.ru/ (дата обращения: 05.01.2024 г.).

4. **Перечень подлежащих разработке вопросов**

4.1. Провести анализ задач ОГЭ.
4.2. Провести анализ способов классификации графиков функций с помощью нейросетевых технологий.

4.3. Подготовить набор данных для обучения нейросетевой модели.
4.4. Реализовать модель и провести ее обучение.
4.5. Провести анализ результатов классификации на данных используя различные параметры нейронной сети.
5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
доцент кафедры СП, к.ф.-м.н. Т.Ю. Маковецкая

Задание принял к исполнению И.М. Сысоева

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

В России с 2000 года стали проводить экспериментальную государственную итоговую аттестацию (ГИА) в девятом классе в некоторых регионах, в том числе и в Челябинской области. В 2009 году единый государственный экзамен (ЕГЭ) в конце одиннадцатого класса стал обязательным. В 2014 году аттестацию для девятого класса переименовали в основной государственный экзамен (ОГЭ) и сделали обязательной, а аббревиатура ГИА стала обозначать совокупность всех аттестаций.

ОГЭ – представляет собой форму государственной итоговой аттестации, проводимой в целях определения соответствия результатов освоения обучающимися основных образовательных программ основного общего образования требованиям федерального государственного образовательного стандарта [22]. Другими словами, ОГЭ – это тестирование, направленное на проверку знаний, полученных в школе за 9 лет обучения, в котором принимают участие все обучающиеся, не имеющие академической задолженности и в полном объеме выполнившие учебный план или индивидуальный учебный план (имеющие годовые отметки по всем учебным предметам учебного плана за 9 класс не ниже удовлетворительных), а также имеющие результат «зачет» за итоговое собеседование по русскому языку.

Трудно переоценить важность данных экзаменов: их результат играет большую роль в итоговых баллах в аттестате. Он также учитывается при поступлении в профильные классы при получении среднего образования и в средние специальные учебные заведения (техникумы, лицеи, колледжи), в том числе и на бюджетных формах.

Всем участникам учебного процесса (учителю, обучающемуся и его родителям) необходимо иметь актуальную информацию об усвоении учебного материала учащимся [4]. В частности, понимать, какие типы заданий, представленных в контрольных измерительных материалах (КИМ), тре-

буют дополнительного изучения. В современном образовательном пространстве актуальность разработки инновационных методов оценки знаний учащихся, организации тематического повторения и подготовки к прохождению итоговой аттестации становится все более важной задачей.

В данном контексте особенно важным является развитие автоматизированных систем, способных эффективно классифицировать задачи, представленные в основном государственном экзамене. Эти задачи не только тестируют понимание материала учащимися, но также предоставляют возможность анализа способов их решения. Некоторые задачи, в частности 11 задание ОГЭ (установление соответствие между графиками функций и формулами, которые их задают), могут быть размножены с помощью замены входных данных и использованы при подготовке учащихся. В связи с этим стоит рассмотреть вопрос генерации подобных задач.

Использование нейронных сетей в образовании позволяет использовать инновационные подходы в обработке информации, такой как изображение графиков функций, изучаемых в основной школе. Разработка инструмента, способного классифицировать задачи по способам решения, является важным как в учебном процессе в целом, так и при подготовке к ОГЭ. Это может быть использовано для персонализации обучения, выявления слабых мест ученика, а также для улучшения общей методики преподавания. Использование нейронных сетей в учебном процессе позволяет создавать индивидуальные программы обучения, анализировать индивидуальные проблемы каждого учащегося и создавать специальные задачи для решения индивидуальных задач.

Выбор темы «Разработка нейронной сети для классификации задач ОГЭ по способам решения» не только отражает современные вызовы в образовательной сфере, но и предлагает инновационный подход, который может привести к улучшению качества образования и подготовке учащихся к применению своих знаний в реальных ситуациях. В результате успешной

реализации проекта ожидается создание инструмента, способного автоматически классифицировать задачи ОГЭ по выбранной теме по способам решения, что в свою очередь может облегчить процесс подготовки учащихся к экзамену и выявления уровня усвоения материала.

Постановка задачи

Цель данной работы заключается в разработке и обучении нейронной сети для классификации задач ОГЭ по способам их решения. Для выполнения заявленной цели требуется решить следующие задачи:

1) выполнить анализ предметной области (проанализировать содержание КИМ по математике и инструментов для подготовки учащихся к ОГЭ);

2) подготовить набор данных для обучения нейронной сети для классификации графиков функций;

3) написать программные коды для генерации графиков;

4) создать нейронную сеть, которая позволит распознавать функции по их графикам (линейная функция, парабола, гипербола);

5) обучить нейронную сеть классификации и провести эксперименты;

6) сохранить нейронную сеть для дальнейшего использовании, например, в приложении.

Структура и содержание работы

Работа состоит из введения, четырех глав, заключения и списка литературы и приложения. Объем работы составляет 55 страниц, объем списка литературы – 27 источников.

В первой главе представлен анализ предметной области. В частности, описывается необходимость изучения контрольных измерительных материалов (КИМ), определяемых на основе федерального государственного образовательного стандарта, для подготовки набора данных и выбора подходящего вида нейронной сети. Затем предоставляется краткое содержание

КИМ по математике, описывается структура ОГЭ по этому предмету, уделяется особое внимание заданию № 11, относящемуся к разделу «Функции». Далее рассматриваются инструменты, помогающие при подготовке к экзамену: сборники, сайты и рассматривается архитектура и предварительный дизайн приложения, в котором может быть использована нейронная сеть. В главе ставятся задачи, необходимые для выполнения квалификационной работы.

Вторая глава посвящена набору данных. В данной главе показан процесс сбора данных из открытых источников. Также описывается процесс привлечения учащихся 11 класса, сдававших ОГЭ по математике после 9 класса для формирования набора данных. Показана необходимость добора данных при помощи генерации графиков функций с использованием языка программирования Python, приведены примеры кодов. Дана классификация данных, для использования в нейронной сети.

В третьей главе представлен обзор различных архитектур нейронных сетей, включая многослойные персептроны (MLP), сверточные нейронные сети (CNN), рекуррентные нейронные сети (RNN), глубокие нейронные сети (DNN) и специализированные архитектуры. В разделе представлены схемы архитектур и их характеристики, а также обсуждаются преимущества и недостатки каждой архитектуры в контексте конкретной задачи.

Далее показана важность программных средств для реализации нейронных сетей и описывает используемые инструменты и библиотеки в данном проекте. TensorFlow выбран в качестве основного инструмента благодаря его гибкости и высокой производительности. Основные библиотеки и модули включают TensorFlow, NumPy, Matplotlib и модуль os. TensorFlow предоставляет высокоуровневый API через Keras для создания модели и обучения нейронной сети. Модули из TensorFlow, такие как Sequential, Conv2D, MaxPooling2D, Flatten и Dense, используются для построения и компиляции модели. Классы ImageDataGenerator и image из библиотеки Keras используются для предобработки изображений перед обучением сети.

NumPy используется для работы с данными и создания массивов, а Matplotlib – для визуализации процесса обучения. Модуль os используется для работы с файловой системой и чтения количества классов из директории с данными.

В главе описывается процесс создания, обучения и оценки нейронной сети для классификации изображений. В рамках проекта используется метод глубокого обучения для распознавания объектов на изображениях с использованием набора данных, содержащего изображения различных классов. Описываются преимущества выбранной архитектуры модели, включая эффективное извлечение признаков, удобство использования Sequential API, настраиваемость и адаптивность оптимизатора и функции потерь. Далее рассматриваются эксперименты, начиная с базовой модели. Сравниваются результаты экспериментов и делаются выводы о влиянии параметров на качество модели. Также, рассматривается альтернатива сверточным нейронным сетям для классификации изображений: полносвязные нейронные сети, дается ее описание, реализация с использованием TensorFlow, описывается обучение результаты сравниваются с CNN.

В приложении А содержится исходный код сверточной нейронной сети.

В приложении Б представлен исходный код полносвязной нейронной сети.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Для разработки нейронной сети для классификации задач ОГЭ по способам решения необходимо изучить содержание контрольных измерительных материалов (КИМ), найти наиболее подходящий для решения поставленной задачи заданий вид нейронных сетей, подготовить набор данных для обучения нейронной сети.

1.1. Содержание КИМ по математике

Порядок проведения ОГЭ определен Федеральным законом от 29.12.2012 г. № 273-ФЗ «Об образовании в Российской Федерации» [5] и «Порядком проведения государственной итоговой аттестации по образовательным программам основного общего образования» [6], утвержденным приказом Минпросвещения России и Рособрнадзора от 04.04.2023 г. № 232/551. Основой содержание КИМ ОГЭ является федеральный государственный образовательный стандарт (ФГОС) основного общего образования.

1. Приказ Министерства просвещения Российской Федерации от 31.05.2021 г. № 287 «Об утверждении федерального государственного образовательного стандарта основного общего образования» [10].

2. Приказ Министерства образования и науки Российской Федерации от 17.12.2010 г. № 1897 (с изменениями 2014–2022 гг.) [11].

3. Письмо Рособрнадзора от 19.12.2023 г. № 04-389 [12].

При разработке КИМ ОГЭ учитывается содержание федеральной образовательной программы основного общего образования (приказ Министерства просвещения Российской Федерации от 18.05.2023 г. № 370 «Об утверждении федеральной образовательной программы основного общего образования») [1522].

ОГЭ по математике состоит из двух частей. Часть 1 содержит 19 заданий с кратким ответом; часть 2 состоит из 6 заданий с развернутым ответом.

Распределение заданий части 1 по разделам содержания курса математики представлено ниже:

1) числа и вычисления;
2) алгебраические выражения;
3) уравнения и неравенства;
4) числовые последовательности;
5) функции;
6) умение использовать свойства последовательностей;
7) координаты на прямой и плоскости;
8) геометрия.

В данной работе рассматривается задание № 11, относящееся к разделу № 5 «Функции». Правильно решенное задание позволяет определить умение строить графики функций, использовать графики для определения свойств процессов и зависимостей, умение выражать формулами зависимости между величинами. Образцы задания приведены ниже.

На рисунке 1 надо установить соответствие между графиками функций и формулами, которые они задают.

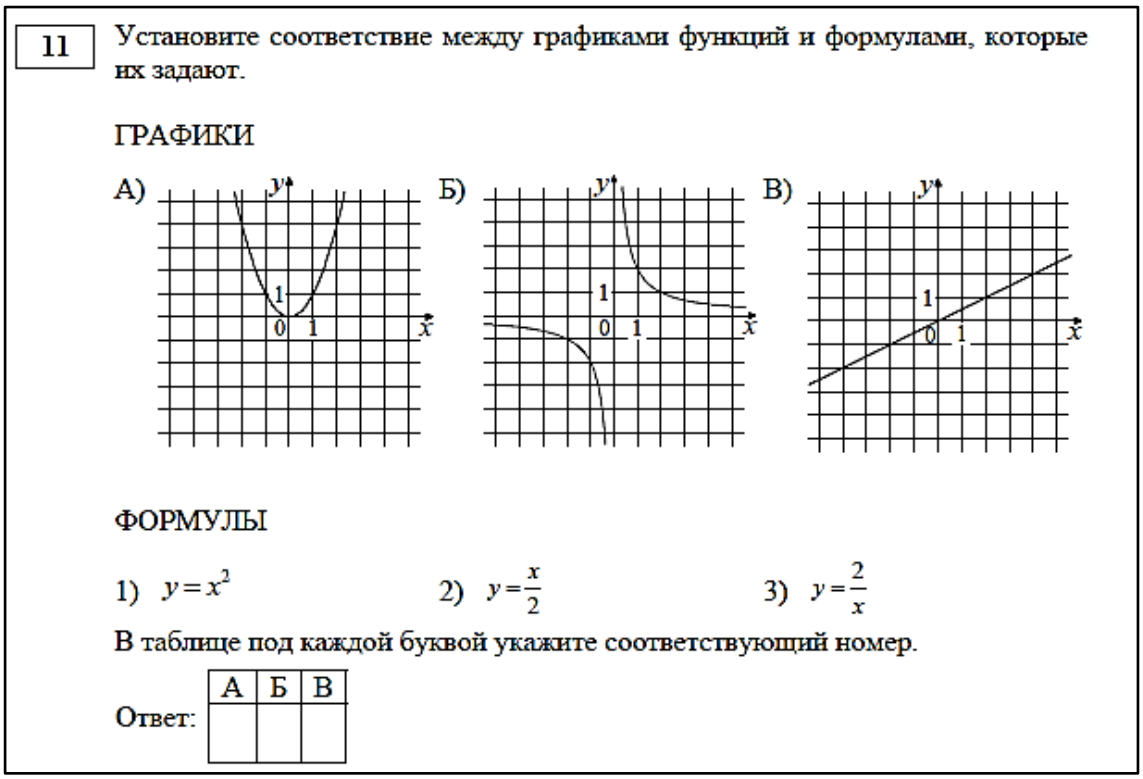

Рисунок 1 – Образец задания 11 (ОГЭ)

На рисунке 2 надо установить соответствие между графиками функций и коэффициентами из формулы квадратичной функции.

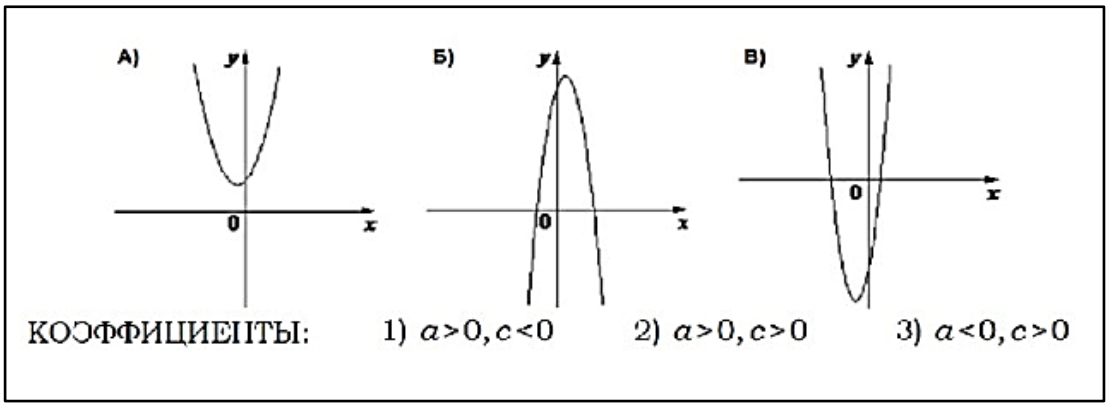

Рисунок 2 – Образец задания 11 (ОГЭ)

На рисунке 3 надо установить соответствие между графиками функций и коэффициентами из формулы линейной функции.

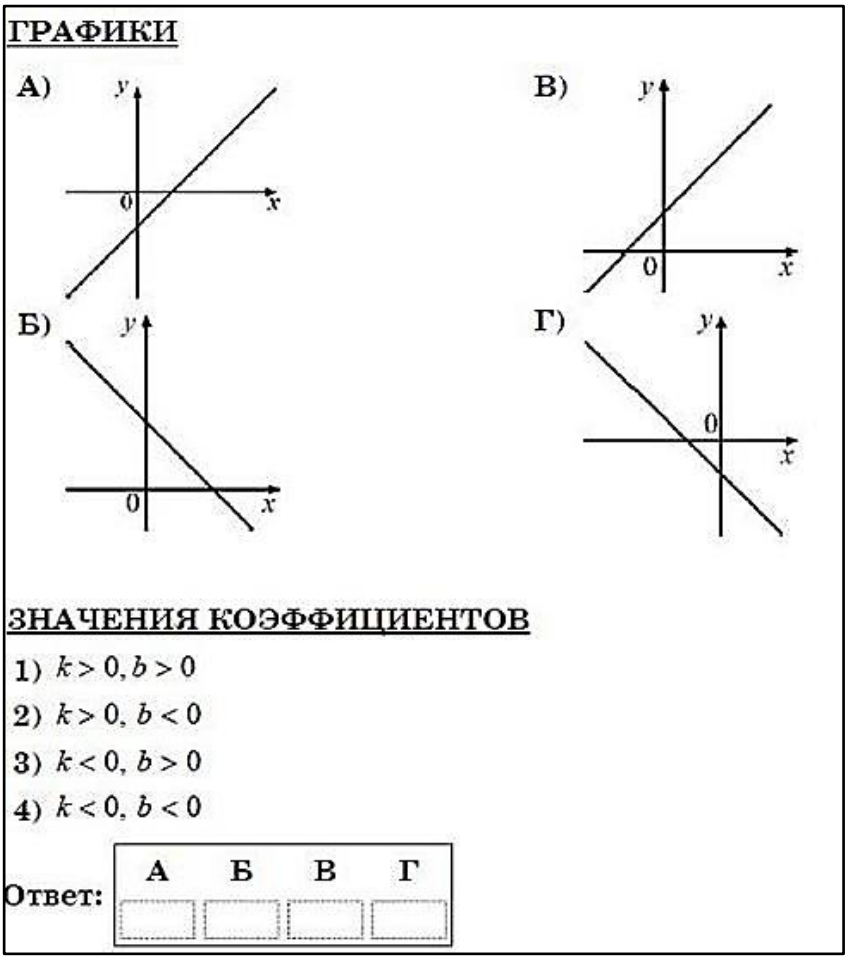

Рисунок 3 – Образец задания 11 (ОГЭ)

1.2. Обзор инструментов для подготовки учащихся

При работе с выпускниками учитель математики ставит перед собой цель: подготовить всех учащихся к успешной сдаче ОГЭ с хорошим качеством. Для этого учителю необходимо обладать необходимыми компетенциями, совершенствовать структуру и содержание учебного материала в ходе подготовки к ОГЭ, систематизировать повторение программного материала, отрабатывать тестовые технологии в ходе работы с контрольно-измерительными материалами через личностно-ориентированный подход.

В настоящее время существует множество различных инструментов, помогающих учителю в этой работе. Познакомится с графиками функций учащиеся могут в учебниках алгебры. При подготовке к экзаменам используются сборники вариантов КИМов. Такие сборники выпускаются под редакцией авторов, выпускающих учебники. Сборники могут содержать различное количество вариантов. Как правило, задания в сборниках однотипны, и не показывают полного разнообразия заданий ОГЭ. Пример такого сборника показан на рисунке 4.

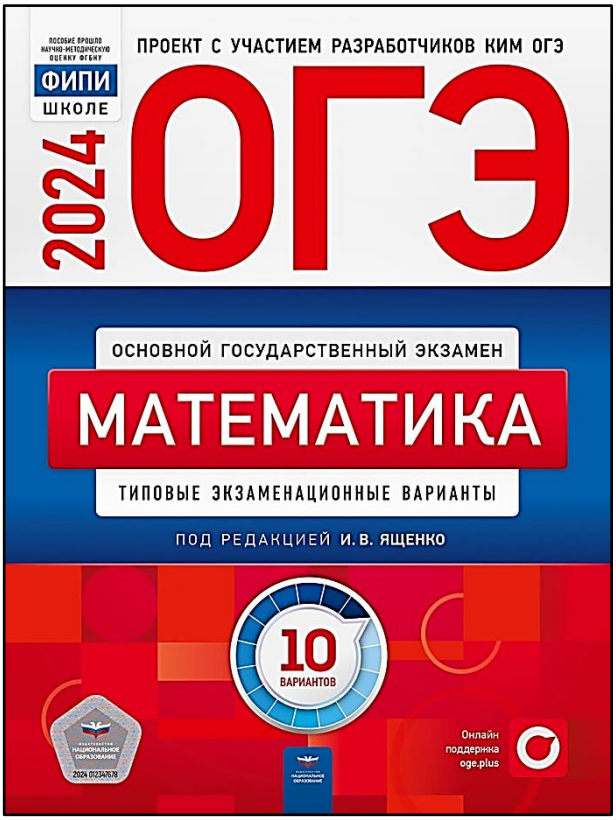

Рисунок 4 – Сборник вариантов ОГЭ по математике 2024

Наиболее популярными при подготовке к экзамену являются сайты. Основой является сайт «ФИПИ» [22]. На данном сайте представлены задания, из которых в дальнейшем формируются КИМы для экзаменов. На данном сайте есть возможность решить все задания и получить обратную связь (верный или неверный ответ). К сожалению, этот ресурс не показывает правильный ответ или правильное решение, если ответ неверный. На рисунке 5 представлен сайт «ФИПИ».

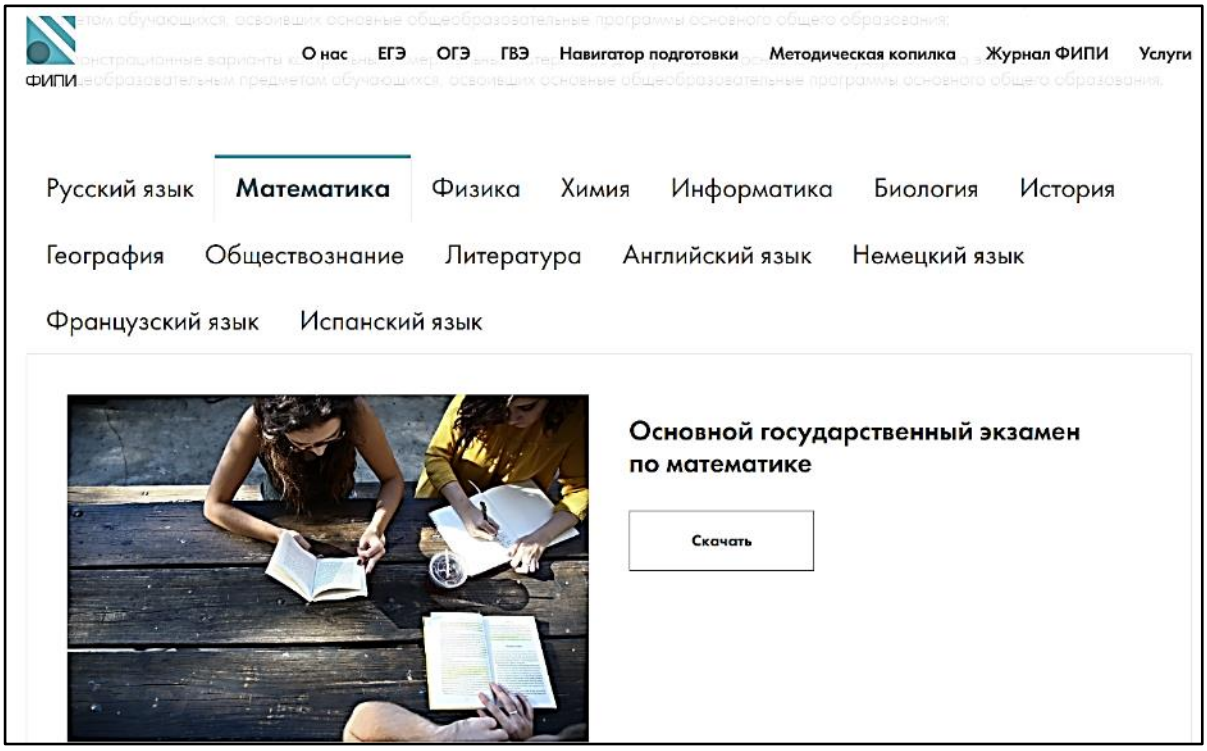

Рисунок 5 – Сайт ФИПИ

Не менее популярен и образовательный портал «СДАМ ГИА» [23]. Этот ресурс дает возможность учащимся не только выполнять задание и проверять правильность его выполнения, но и показывает решение всех заданий. Но, задания, представленные на данном портале, не всегда актуальны. Задания на сайт могут предлагать пользователи сайта. Примеры решения, в большинстве случаев, не соответствуют стандартам оформления экзаменационных работ, что может привести к потере баллов на экзамене. На рисунке 6 представлен сайт «СДАМ ГИА».

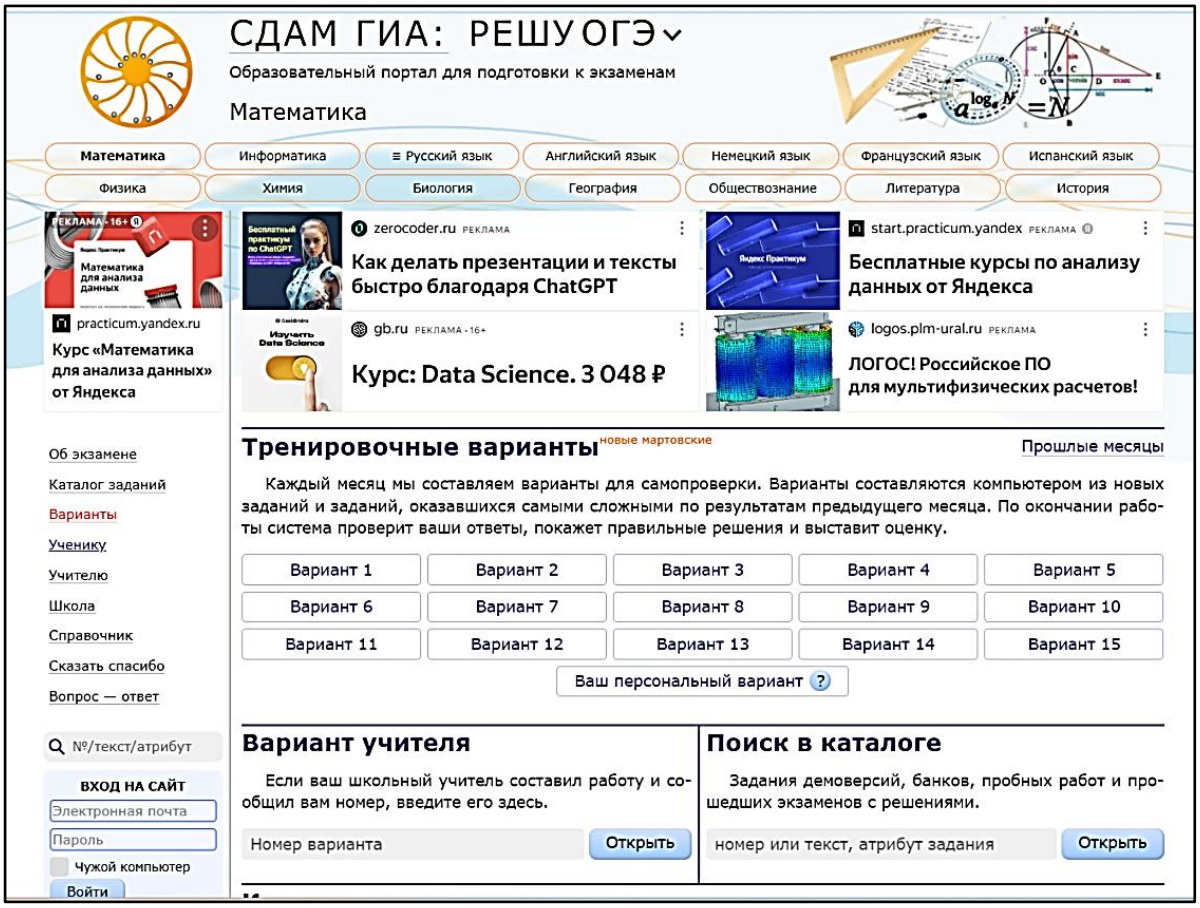

Рисунок 6 – Образовательный портал «СДАМ ГИА»

Одна из проблем, с которой сталкивается учитель математики заключается в том, что задания ОГЭ и задачи из учебников в большинстве случаев почти никак не связаны. Следовательно, учитель вынужден по каждой теме подбирать задания из ОГЭ для включения их в перечень решаемых задач на уроке, чтобы дети могли знакомиться с разными формулировками и видеть все разнообразие математических примеров. Следует также отметить, что перечисленные выше инструменты предоставляют малое разнообразие заданий на определенную тему: как правило рассматриваются задания, представленные на сайте ФИПИ. Например, на сайте федеральной службы по надзору в сфере образования и науки в разделе «Спецификация КИМ ОГЭ 2024» [22] утверждается, что учащиеся должны уметь решать задачи разных типов (в том числе на проценты, доли и части, движение, работу, цену това-

ров и стоимость покупок и услуг, налоги, задачи из области управления личными и семейными финансами), составлять выражения, уравнения, неравенства и системы по условию задачи, исследовать полученное решение и оценивать правдоподобность полученных результатов. При этом, задачи изучаются в 8 классе в разделе дробно-рациональные уравнения и представлены одной или двумя задачами каждого типа. Как правило, именно эти задачи предлагаются во всех сборниках для подготовки к экзамену. Но для отработки темы необходимо решать большое количество задач по каждому типу.

В связи с этим, существует необходимость в генерации дополнительных заданий для подготовки учащихся к сдаче ОГЭ.

1.3. Постановка задач для разработки нейронной сети

Целью выпускной квалификационной работы является разработка нейронной сети для классификации задач ОГЭ по способам решения. Для написания нейронной сети необходимо решить следующие задачи.

1. Сбор набора данных и его подготовка для использования в нейронной сети.

2. Выбор оптимальной архитектуры нейронной сети для задачи классификации и разработка методологии обучения модели с учетом специфики данных ОГЭ и требований задач.

3. Проектирование и реализация архитектуры нейронной сети и обучение модели на обучающей выборке с последующей валидацией и тестированием.

4. Анализ результатов классификации на данных и сравнение существующих методов классификации, и оценка преимуществ предложенной модели.

5. Анализ влияния различных параметров нейронной сети на ее производительность и оптимизация параметров для достижения лучших результатов.

Данные требования представлены на рисунке 7.

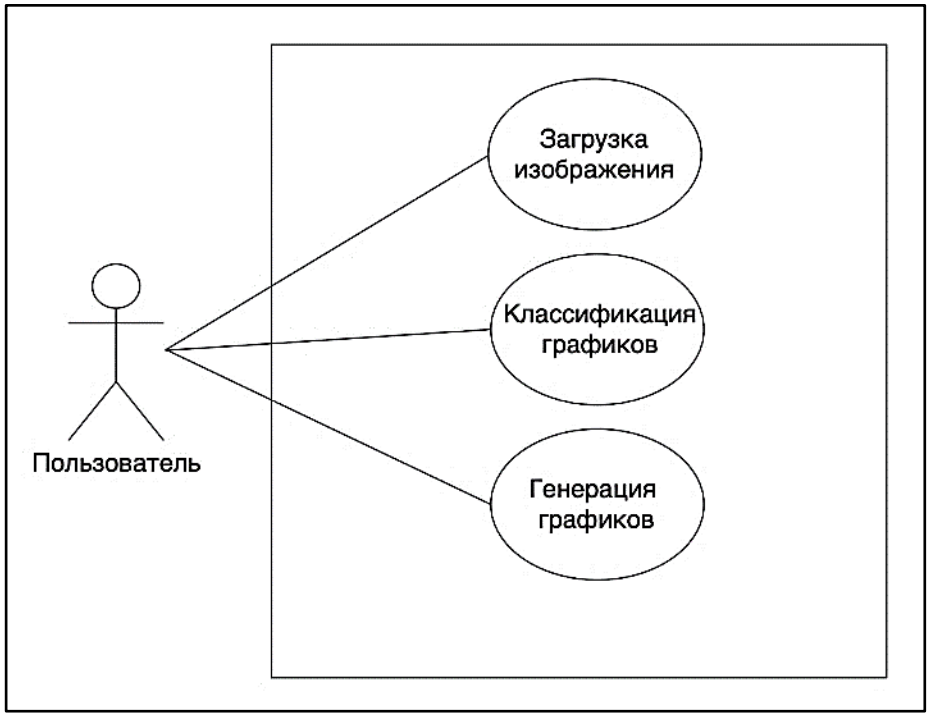

Рисунок 7 – Диаграмма вариантов использования

1.4. Архитектура приложения

Примером использования нейронной сети для классификации задач ОГЭ по способам решения может служить веб-приложение для генерации и классификации графиков функций. Этот инструмент предоставляет уникальную возможность загружать изображения графиков функций и получать подробное объяснение их свойств.

Для этого можно создать приложение, которое выполняет следующие шаги:

1) генерировать случайный график функции;
2) отображать график пользователю;
3) предлагать варианты ответа пользователю (классификация графика);
4) проверять правильность ответа;
5) выдавать новое задание заданного класса по запросу пользователя.

Архитектуру такого приложения могут представлять несколько компонентов, которые взаимодействуют между собой для реализации требуемого функционала.

1. Обработчик изображений – блок, в котором данные изображений обрабатываются, и передаются в компонент моделей нейросети. Также происходит генерация изображений для передачи веб приложению.
2. Нейронная сеть – компонент обученной модели нейронной сети.
3. Веб-приложение – компонент, отвечающий за организацию передачи данных конечному пользователю, и выполнение операции для удовлетворения клиентских запросов.

Графический интерфейс состоит из нескольких веб страниц. Главная страница приложения может выглядеть так, как представлено на рисунке 8.

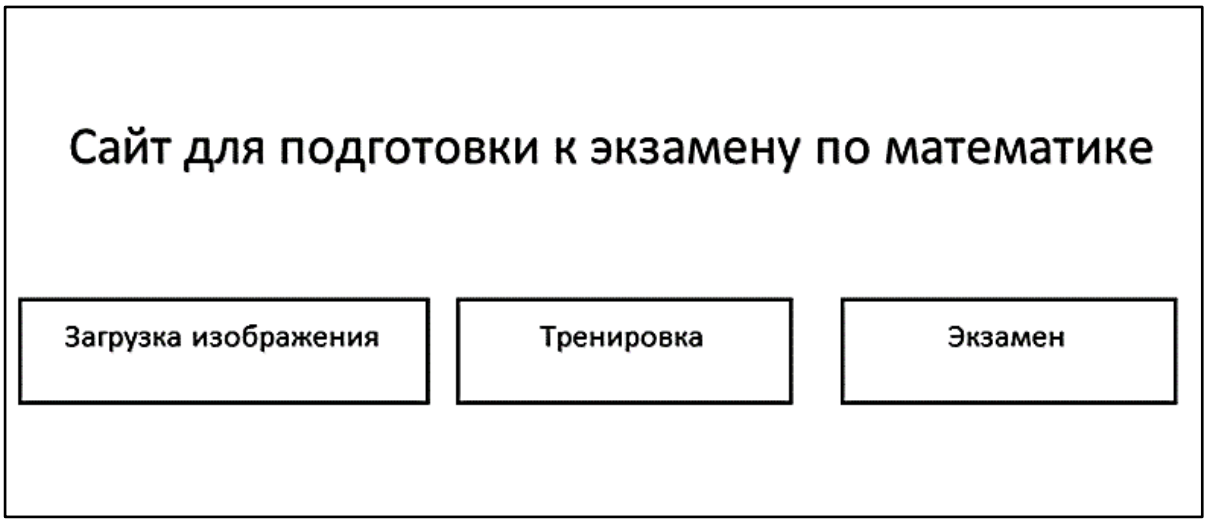

Рисунок 8 – Главная страница приложения

Из вышесказанного видно, что для создания такого приложения необходим набор данных с графиками функций и их классификация, а также обученная нейронная сеть для классификации графиков.

2. НАБОР ДАННЫХ

Задача классификации объектов является одной из важнейшей в области применения нейронной сети. Для решения данной задачи можно подобрать различные алгоритмы реализации. Однако, в первую очередь необходимо подготовить данных для обучения [2]. Данные можно собрать из открытых источников и разбить их на классы. Если данных для обучения нейронной сети недостаточно, то их надо сгенерировать.

2.1. Сбор данных

Сбор данных из открытых источников

Собранные данные должны полностью соответствовать выбранной модели нейронной сети и предусматривать все возможные сценарии.

В нашем случае, сбор изображений графиков функций для их классификации с использованием нейронной сети производился из открытых источников: сайты [22, 23], сборники для подготовки к ОГЭ [24], учебники по алгебре. Данные были сохранены в формате .png.

Ожидаемо, возникла проблема повторяемости материала во всех источниках. Более того, изображений, собранных из открытых источников, оказалось мало для обучения нейронной сети. Примеры изображений представлены на рисунке 9.

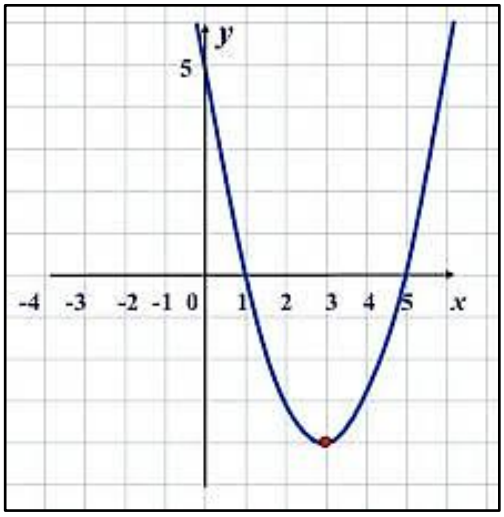

Рисунок 9 – Изображение из открытого источника

Обзор данных, собранных учащимися

Планируется, что нейронная сеть, созданная в рамках данной работы, будет использована в веб-приложении, для подготовки обучающихся к ОГЭ по математике. Поэтому, в ходе сбора данных для тестовой и обучающей выборки было решено обратиться к учащимся 11 класса, которые по окончании 9 класса сдавали ОГЭ, и, следовательно, понимали требования к рисункам для набора данных. Учащимся было предложено найти в интернете, сканировать из бумажных источников, нарисовать вручную или используя компьютерные программы или сгенерировать графики функций, согласно тематике 11 задания ОГЭ 9 класса. Выпускники охотно включились в работу. Вариант сканирования бумажных источников и поиск готовых изображений ребят не заинтересовал. В основном, на уроке математике, ребята нарисовали графики функций на тетрадных листах в клетку и листах, формата А4. Некоторые ребята попробовали написать код, для генерации графиков, кто-то построил графики в Microsoft Paint. На рисунке 10 представлены образцы работ на бумаге.

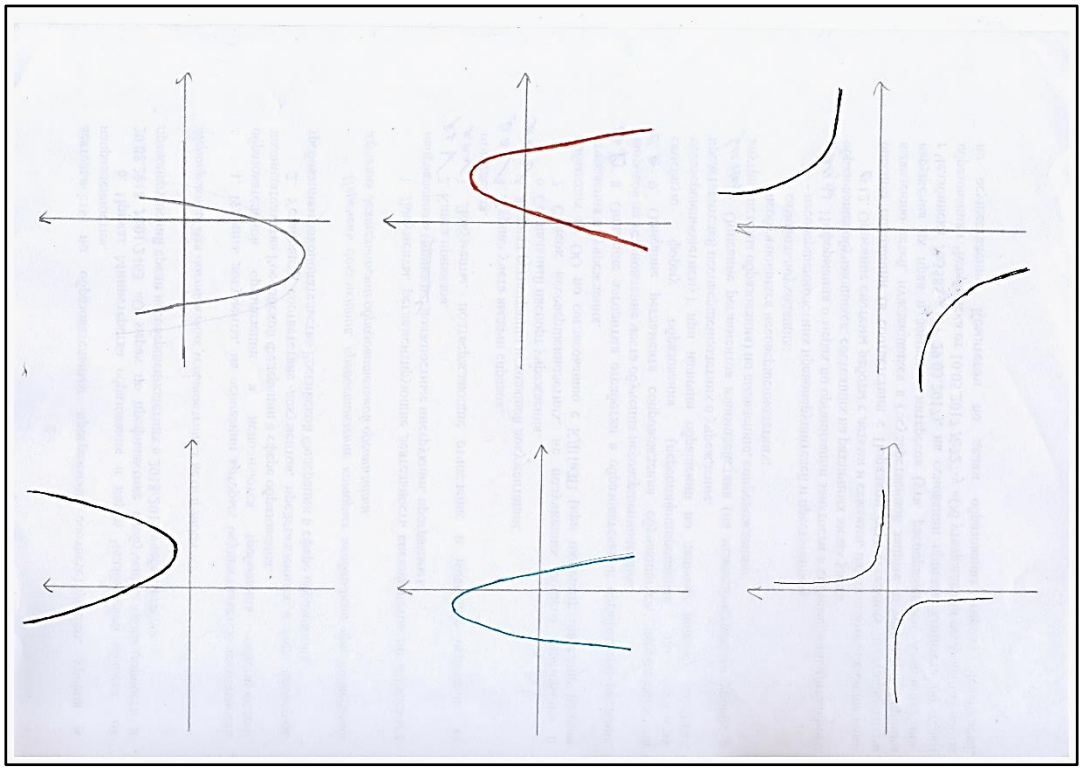

Рисунок 10 – Графики функций, выполненные на бумаге

На рисунке 11 представлен образец кода для генерации параболы, выполненный учащимся.

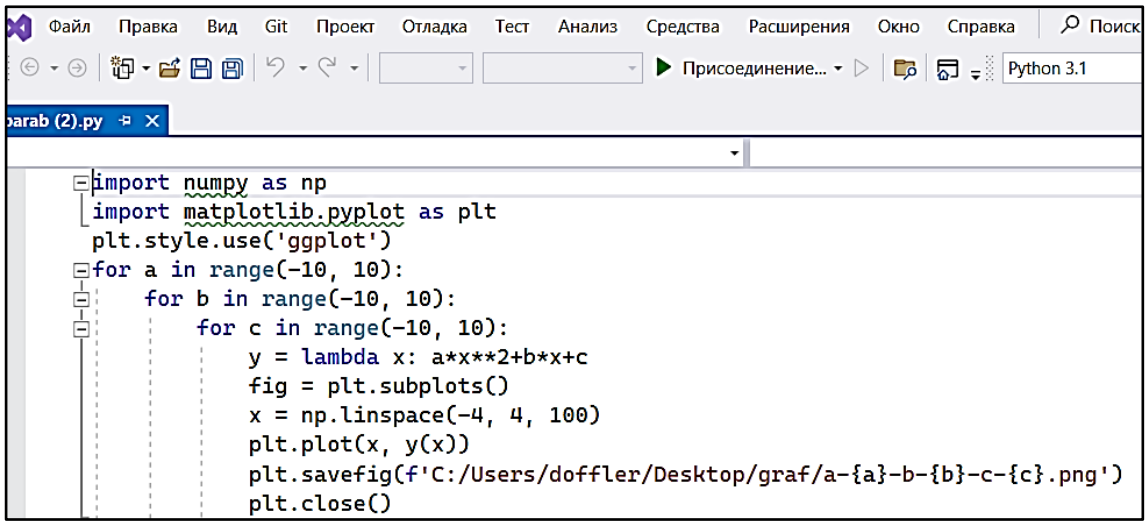

Рисунок 11 – Кода для генерации параболы

На рисунке 12 представлен график, выполненный в программе Paint.

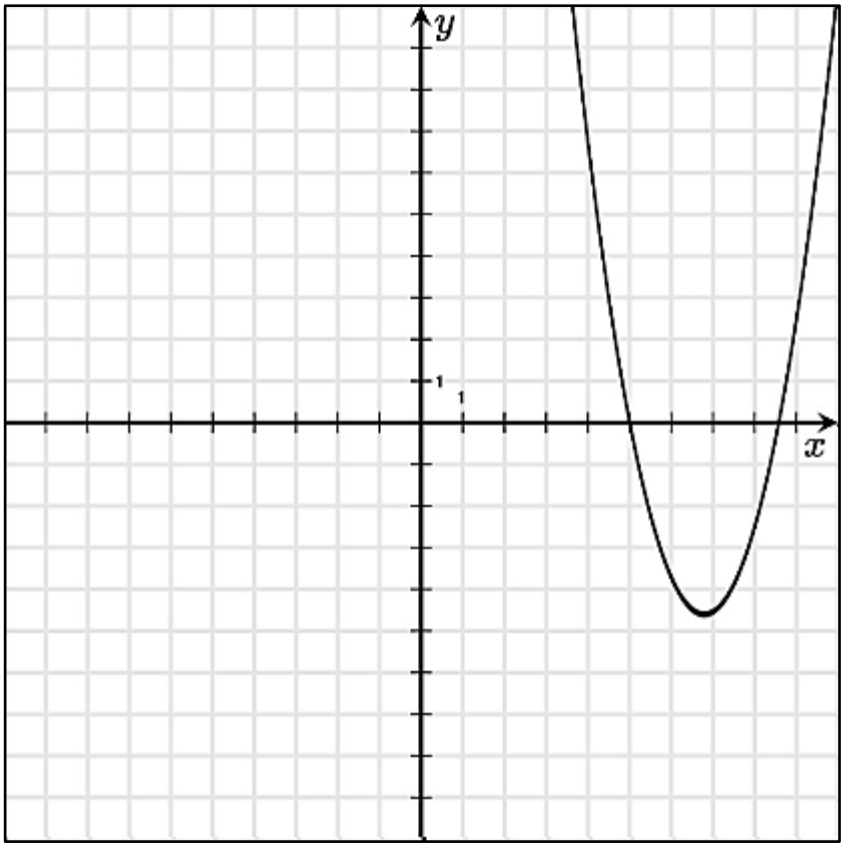

Рисунок 12 – График функции, выполненный в программе Paint

Учащиеся предоставили около 600 изображений.

2.2. Классификация данных

Данных после сбора необходимо структурировать. Набор данных состоит из графиков четырех типов функций: линейная функция (график представляет собой прямую линию), квадратичная функция (график представляет собой параболу), функция обратной пропорциональности (график представляет собой гиперболу) и функция корня (представляет собой кривую, которая имеет точку перегиба). В свою очередь, прямая, парабола и гипербола в зависимости от коэффициентов обладают различными характерными свойствами и их графики отражают эти зависимости. К примеру, линейная функция $y = kx + b$ при $k > 0$ возрастает. А график гиперболы при $k > 0$ расположен в I и III четвертях координатной плоскости. Графики параболы и гиперболы можно выделить в 4 класса, линейной функции в 5 классов и для графиков функций, не относящихся к вышеперечисленным, сформируем класс Other. Принимая это во внимание, собранные данные были распределены по классам:

1) hyperbola_1: $k > 1$;
2) hyperbola_2: $k < -1$;
3) hyperbola_3: $0 < k < 1$;
4) hyperbola_4: $-1 < k < 0$;
5) liner_1: $k > 0, b > 0$;
6) liner_2: $k < 0, b < 0$;
7) liner_3: $k < 0, b > 0$;
8) liner_4: $k > 0, b < 0$;
9) liner_4: $k = 0$;
10) other;
11) sqr_1: $a > 0, c > 0$;
12) sqr_2: $a < 0, c < 0$;
13) sqr_3: $a > 0, c < 0$;
14) sqr_4: $a < 0, c > 0$.

Анализ собранных данных показал, что каждый класс содержит от 10 до 30 изображений. В связи с недостаточным количеством собранных таким способом данных для обучения нейронной сети возникла необходимость генерации изображений.

2.3. Генерация данных

Генерация графиков была выполнена на языке программирования Python. При выполнении данной части работы учитывались следующие параметры: наличие или отсутствие координатной сетки, толщина и цветность как самого графика, так и координатных осей, размерность рисунка в координатах. В листинге 1 приведен пример генерации графиков обратной пропорциональности (гиперболы) при $k > 1$. Аналогичные коды были написаны для других классов обратной пропорциональности в зависимости от значений коэффициента k.

Листинг 1 – Гипербола, при $k > 1$

```
def linear_graph_one(k, filename):
    x = np.arange(-10, 11) # Генерируем значения x от -10 до 10
    y = [k/i for i in x] # Вычисляем значения y
    plt.figure(fig size=(10, 10))
    plt.plot(x, y, c="r", lw =5) # Рисуем график
    plt.axvline (x=0, c="k", lw =4)
    plt.axhline (y=0, c="k", lw =4)
    plt.title('График функции') # Подписываем заголовок
    plt.grid(False) # Отображение сетки
    plt.savefig(filename) # Сохраняем график в файле
    plt.close() # Закрываем текущую фигуру
# Генерируем графики
for i in range(15):
    filename = f'1{i}.png' # Генерируем имя файла для каждого графика
    k = random.uniform(50, 100) # Генерируем случайный коэффициент k
    linear_graph_one (k, filename)
```

В отличии от квадратичной функции и обратной пропорциональности, которые разделены на 4 класса, линейная функция разделена на 5 классов: в отдельный класс выделен коэффициент $k = 0$. Определенные параметры, определяющие вид соответствующего графика, оставались неизменными на 15 – 20 изображениях. Листинг 2 показывает генерацию графиков линейной функции при $k > 0, b > 0$.

Листинг 2 – Линейная функция, при $k > 0, b > 0$

```
def linear_graph_one(k, b, filename):
    x = np.arange(-10, 11, 1) # Генерируем значения x от -10 до 10
    y = [k * i + b for i in x] # Вычисляем значения y
    plt.figure(figsize=(10, 5))
    plt.plot(x, y, c="r") # Рисуем график
    plt.axvline (x=0, c="b")
    plt.axhline (y=0, c="b")
    plt.title('График функции’) # Подписываем заголовок
    plt.grid(True) # Включаем отображение сетки (False)
    plt.savefig(filename) # Сохраняем график в файле
    plt.close() # Закрываем текущую фигуру
# Генерируем 15 графиков
for i in range(15):
    filename = f'linear_graph_one{i}.png’ # Генерируем имя файладлярафика
    k = random.uniform(0, 10) # Генерируем случайный коэффициент k
    b = random.uniform(0, 55) # Генерируем случайный коэффициент b
    linear_graph_one (k, b, filename)
```

Аналогичные требования были выполнены и при построении графиков квадратичной функции. На листинге 3 показан код генерации графиков параболы $a < 0, c > 0$.

Листинг 3 – Квадратичная функция

```
def generate_parabola_graph(a, b, c, filename):
    x = np.arange(-10, 10) # Генерируем значения x от -10 до 10
    y = [a * (i  2) + b * i + c for i in x] # Вычисляем значения y

    plt.figure(figsize=(15, 15))
    plt.plot(x, y, c="c", lw=4) # Рисуем график
    plt.axvline (x=0, c="k", lw=2)
    plt.axhline (y=0, c="k", lw=2)
    plt.title('График') # Подписываем заголовок
    plt.grid(False) # Включаем отображение сетки
    plt.savefig(filename) # Сохраняем график в файле
    plt.close() # Закрываем текущую фигуру

# Генерируем 15 графиков квадратичной функции
for i in range(15):
    filename = f'parabola2graph_four{i}.png' # Генерируем имя файла для
каждого графика
    a = random.uniform(-10, 0.1) # Генерируем случайный коэффициент a
    b = random.uniform(-5, 5) # Генерируем случайный коэффициент b
    c = random.uniform(20, 100) # Генерируем случайный коэффициент c
    generate_parabola_graph (a, b, c, filename)
```

Графики функций, не относящиеся к вышеперечисленным, были отнесены к классу Other. Данный класс содержит графики таких функций, как кубическая парабола, функции под знаком модуля, функция квадратного корня. Листинг 4 демонстрирует код генерации графика функции квадратного корня.

Листинг 4 – Функция квадратного корня

```
def linear_graph_one(k, filename):
    x = np.arange(0, 55) # Генерируем значения x от -10 до 10
    y = [k*math.sqrt (i) for i in x]

   plt.figure(figsize= (40, 25))
   plt.plot(x, y, c="c", lw=6) # Рисуем график
   plt.axvline(x=0, c="k", lw=6)
   plt.axhline(y=0, c="k", lw=6)
   plt.title('График функции') # Подписываем заголовок
   plt.grid(True) # Включаем отображение сетки
   plt.savefig(filename) # Сохраняем график в файле
   plt.close() # Закрываем текущую фигуру

# Генерируем 15 графиков
for i in range(15):
   filename = f'14other_{i}.png' # Генерируем имя файла для каждого графика
   k = random.uniform(-1, 1) # Генерируем случайный коэффициент k
   linear_graph_one (k, filename)
```

Сгенерированные графики функций были сохранены в формате .png в папках с соответствующими классами. На рисунке 13 показан образец генерированного графика гиперболы.

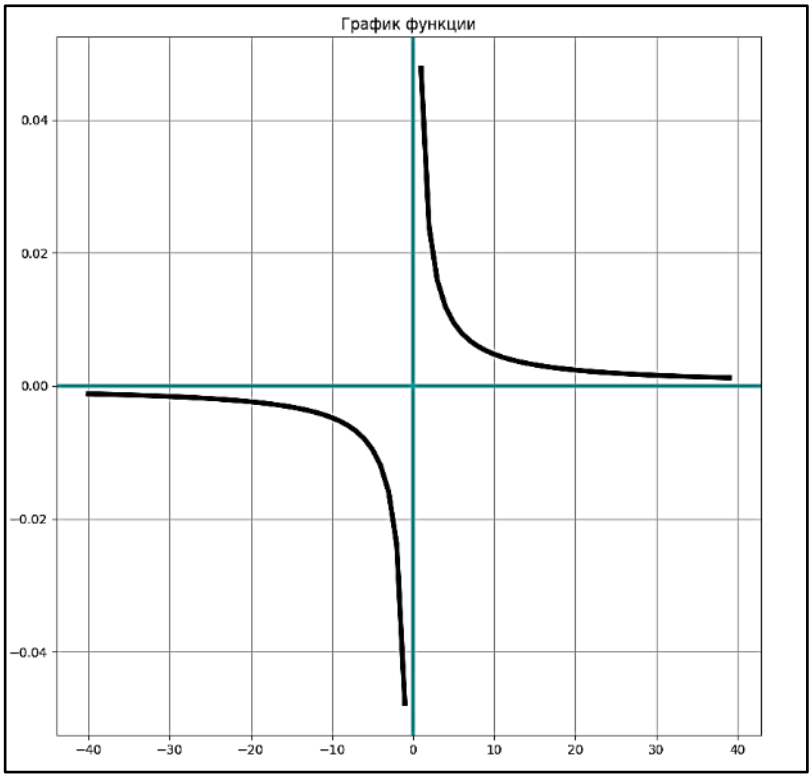

Рисунок 13 – График гиперболы

Набор данных, собранный всеми перечисленными способами, составляет 4200 изображений.

3. НЕЙРОСЕТЕВАЯ МОДЕЛЬ

Нейронные сети (НС) – вычислительные модели, вдохновлены биологическими нейронными сетями человеческого мозга. Они используются для обработки данных и выполнения сложных задач, таких как распознавание образов, классификация, прогнозирование и многое другое.

Обучение нейронных сетей происходит путем настройки различных параметров модели на основе обучающих данных. Это может быть выполнено с использованием различных методов, таких как обратное распространение ошибки, стохастический градиентный спуск и другие.

Нейронные сети обладают большой мощностью. Они способности решать и сложные задачи, и автоматизировать процессы. Благодаря адаптации к различным типам задач, нейронные сети широко используются в таких областях, как компьютерное зрение, обработка естественного языка, образование, медицина, финансы, промышленность и многое другое. Работу нейронных сетей, несмотря на их распространенность, многие сравнивают с «черными ящиками»: трудно интерпретировать их работу, обучение нейронных сетей требует больших объемов данных, высокой вычислительной мощности и времени [25]. Сложность моделей может привести к переобучению на обучающих данных и плохой обобщающей способности на новых данных.

3.1. Топология нейронной сети

Нейронные сети имеют различные архитектуры: многослойные персептроны (MLP), сверточные нейронные сети (CNN), рекуррентные нейронные сети (RNN) и глубокие нейронные сети (DNN) [6].

Многослойные персептроны (MLP) используются в широком спектре задач, включая классификацию, регрессию, распознавание образов и прогнозирование. Они могут использоваться в финансовых приложениях, медицине, обработке естественного языка и других областях. Схема представлена на рисунке 14.

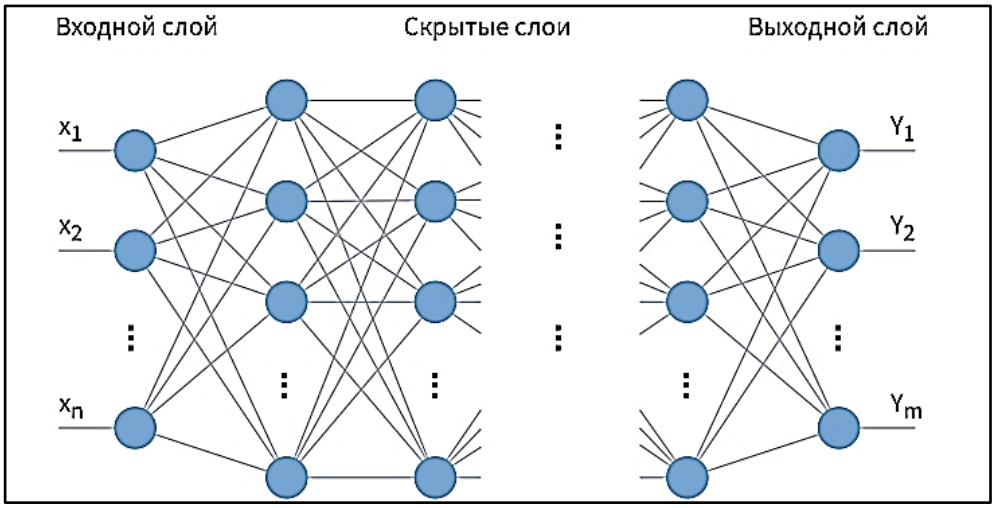

Рисунок 14 – Схема MLP

Сверточные нейронные сети (CNN) эффективно работают с визуальными данными, такими как изображения и видео. Они используются для задач компьютерного зрения, таких как классификация изображений, обнаружение объектов, сегментация и распознавание лиц. Схема представлена на рисунке 15.

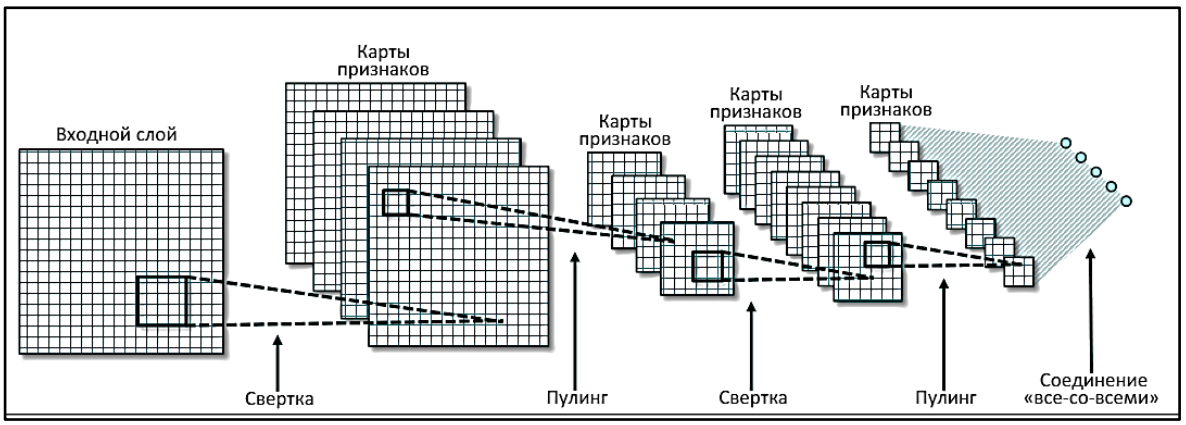

Рисунок 15 – Схема CNN

Рекуррентные нейронные сети (RNN) подходят для обработки последовательных данных, таких как временные ряды, тексты, аудио и видео с переменной длиной. Они широко используются для задач машинного перевода, анализа текста, распознавания речи, генерации текста и других приложений, где важно учитывать контекст. Схема представлена на рисунке 16.

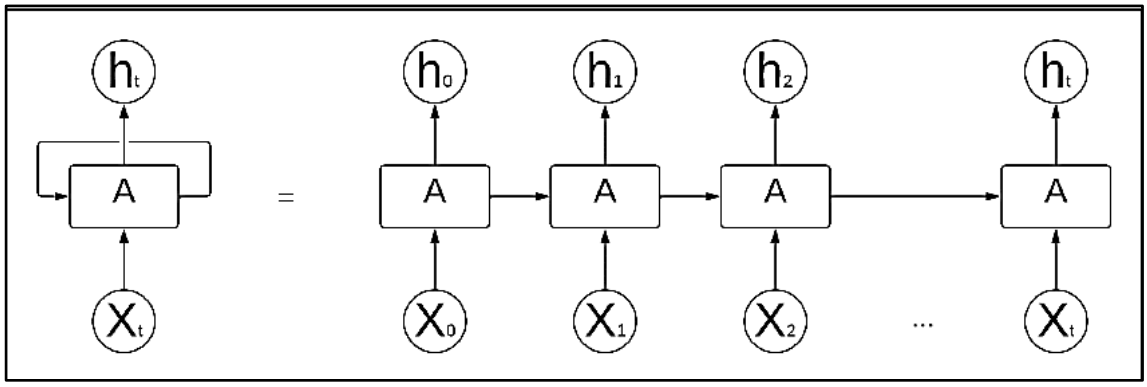

Рисунок 16 – Схема RNN

Долгая краткосрочная память (Long short-term memory; LSTM) – особая разновидность архитектуры рекуррентных нейронных сетей, способная к обучению долговременным зависимостям.

Глубокие нейронные сети (DNN) применяются в различных областях, включая обработку изображений, обработку естественного языка, рекомендательные системы, медицину, финансы и т.д. Они обладают способностью автоматически извлекать признаки из данных и представляют мощный инструмент для обучения на больших объемах данных.

В последние годы было предложено множество специализированных архитектур нейронных сетей для конкретных задач, таких как трансформеры для машинного перевода, генеративно-состязательные сети (GAN) для генерации изображений, а также различные модели для обработки звука, текста и других типов данных.

Для классификации графиков функций в математике можно использовать различные нейронные сети в зависимости от природы данных и требуемой точности классификации. RNN могут быть полезными для анализа временных рядов или последовательностей данных, таких как изменение функции с течением времени. Сверточные нейронные сети широко используются для обработки изображений. CNN могут быть эффективными, если

важны локальные особенности графиков, такие как форма кривых или расположение экстремумов. Выбор сверточных нейронных сетей обусловлен с особенностями их работы.

1. Сверточные слои в нейронной сети способны учитывать пространственные зависимости между пикселями изображения, что позволяет им эффективно извлекать признаки из изображений.

2. В CNN веса сверточных фильтров являются общими для всех частей изображения, что позволяет модели уменьшить количество обучаемых параметров и повысить устойчивость к переобучению.

3. Слои пулинга позволяют понижать размерность данных, сохраняя важные признаки.

4. Многие предобученные сверточные нейронные сети доступны в библиотеках машинного обучения (например, TensorFlow), что упрощает использование уже настроенных моделей для задач классификации и обнаружения объектов.

3.2. Программные средства реализации

Программные средства реализации нейронных сетей является важным инструментом для создания, обучения и развертывания искусственных нейронных сетей. Программное обеспечение облегчает задачу создания нейронной сети. В данном проекте использовались следующие библиотеки и модули: TensorFlow , NumPy, Matplotlib, модуль os.

Открытая библиотека глубокого обучения TensorFlow [27] разработанна командой Google Brain. Она предоставляет мощные инструменты для создания и обучения нейронных сетей. TensorFlow имеет гибкую архитектуру и поддерживает различные платформы, что делает его одним из наиболее популярных фреймворков для глубокого обучения. TensorFlow предоставляет высокоуровневый API для работы с нейронными сетями через

Keras [8]. В данном проекте TensorFlow был выбран как основной инструмент для реализации нейронной сети благодаря своей гибкости, высокой производительности и обширному сообществу.

При создании нейронной сети были использованы следующие модули и классы из TensorFlow [8]:

1) tensorflow.keras.models.Sequential для построения последовательной модели нейронной сети;

2) tensorflow.keras.layers.Conv2D для создания сверточных слоев;

3) tensorflow.keras.layers.MaxPooling2D для применения операции максимального пулинга;

4) tensorflow.keras.layers.Flatten для преобразования данных перед передачей в полносвязные слои;

5) tensorflow.keras.layers.Dense для добавления полносвязных слоев.

Следующие классы ImageDataGenerator и image из библиотеки Keras используются для предобработки изображений:

1) tensorflow.keras.preprocessing.image.ImageDataGenerator позволяет применять различные преобразования к изображениям перед обучением нейронной сети;

2) tensorflow.keras.preprocessing.image.image предоставляет инструменты для работы с изображениями в формате PIL, включая загрузку и преобразование изображений.

Библиотека NumPy языка программирования Python предназначена для работы с массивами и матрицами, а также для обработки данных. В данном проекте NumPy использовался для работы с данными и создания массива num_classes на основе количества папок с изображениями данных.

Библиотека Matplotlib [12] используется для визуализации данных. Она предоставляет широкие возможности для создания различных видов графиков, диаграмм и изображений. Matplotlib является одним из стандартных инструментов для визуализации данных в Python. В данном проекте

Matplotlib использовался для визуализации процесса обучения нейронной сети, включая графики точности и потерь.

Модуль os в Python предоставляет функции для работы с операционной системой. Он позволяет взаимодействовать с файловой системой, выполнять операции с директориями и файлами. В данном проекте модуль os использовался для чтения количества классов из директории с данными.

3.3. Реализация и обучение модели нейронной сети

Рассмотрим процесс создания, обучения и оценки нейронной сети для классификации изображений в рамках проекта. Для создания нейронной сети остановимся на методе глубокого обучения для решения задачи распознавания объектов на изображениях, используя набор данных, содержащий изображения различных классов.

После импорта библиотек и классов загружаем изображения из указанной директории. В проекте данные организованы в виде структуры каталогов, где каждая поддиректория представляла собой отдельный класс изображений. Этот этап показан на листинге 5.

Листинг 5 – Загрузка изображений

```
# Папка с изображениями
data_dir = '/content/drive/MyDrive/DIPLOM/dataset'
# Путь к директории с валидационными данными
validation_data_dir = '/content/drive/MyDrive/DIPLOM/test'
```

В ходе предварительной обработки данных к обучению нейронной сети был создан генератор изображений с использованием класса ImageDataGenerator из библиотеки Keras. Размеры изображений были приведены к одному размеру, определен размер мини пакета. Данные параметры оказывают большое влияние на результативность нейронной сети. Далее формируем генерацию данных, в ходе которой выбирается коэффициент масштабирования для нормализации пиксельных значений (`rescale`) и режим классификации (`class_mode`). Данные перемешиваются для лучшего обучения нейронной сети. Данный этап показан на листинге 6.

Листинг 6 – Размеры изображений

```
# Размеры изображений и размер мини-пакета
img_width, img_height = 28, 28
batch_size = 32

# Создание генератора данных
datagen = ImageDataGenerator(rescale=1. / 255)

# Создание генератора изображений из папки
generator = datagen.flow_from_directory(
    data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=True  # Перемешивание данных
)
```

Нейронная сеть была построена с использованием последовательной модели (Sequential API) из библиотеки Keras. Слои нейронной сети были добавлены последовательно друг за другом. Нейронная сеть состоит из следующих слоев.

1. Сверточный слой (Conv2D) с 32 фильтрами размером 3x3 и функцией активации ReLU. Сверточные слои создают сверточные фильтры, которые проходят по изображению для извлечения признаков. Каждый сверточный слой имеет несколько фильтров, каждый из которых создает карту признаков.

2. Слой максимального пулинга (MaxPooling2D) с размером пула 2x2. Слои пулинг уменьшают размер карт признаков, удаляя избыточную информацию и снижая вычислительную сложность модели.

3. Сверточный слой (Conv2D) с 64 фильтрами размером 3x3 и функцией активации ReLU.

4. Слой максимального пулинга (MaxPooling2D) с размером пула 2x2.

5. Плоский слой (Flatten). Плоский слой преобразует многомерные карты признаков в одномерный вектор, который может быть передан на вход полносвязным слоям.

6. Полносвязный слой (Dense) с 64 нейронами и функцией активации ReLU. Полносвязные слои соединяют каждый нейрон предыдущего слоя с

каждым нейроном текущего слоя. Они выполняют классификацию путем агрегирования информации и вычисления выходного значения для каждого класса.

7. Выходной слой (Dense) с количеством нейронов, соответствующим количеству классов, и функцией активации softmax. В данной нейронной сети для скрытых слоев используется функция активации ReLU (Rectified Linear Unit), а для выходного слоя используется функция активации softmax, которая преобразует выходные значения в вероятности принадлежности к каждому классу.

Создание модели представлено в листинге 7.

Листинг 7 – Создание модели

```
# Создание модели
model = Sequential([
    Conv2D(32, (3, 3), input_shape=(img_width, img_height, 3), activa-
tion='relu'),
    MaxPooling2D(pool_size=pool_size),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=pool_size),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(num_classes, activation='softmax')
])
```

После определения архитектуры нейронной сети модель была скомпилирована с использованием оптимизатора adam (стохастический градиентный спуск с адаптивным коэффициентом обучения) и функции потерь.

В ходе обучение нейронной сети параметры настраиваются посредством моделирования среды, в которую эта сеть встроена. Тип обучения определяется способом подстройки параметров. Различаются алгоритмы обучения с учителем и без учителя. Процесс обучения с учителем представляет собой предъявление сети выборки обучающих примеров. Каждый образец подается на входы сети, затем проходит обработку внутри структуры НС, вычисляется выходной сигнал сети, который сравнивается с соответствующим значением целевого вектора, представляющего собой требуемый выход сети. Для того, чтобы нейронная сети была способна выполнить поставленную задачу, ее необходимо обучить.

Модель была обучена на тренировочных данных с использованием метода `fit` с указанием генератора данных, количества эпох и размера мини-пакета. В процессе обучения модели веса обновлялись с целью минимизации функции потерь. Данный метод представлен в листинге 8.

Листинг 8 – Метод `fit`

```
# Обучение модели
history = model.fit(generator, steps_per_epoch=len(generator), epochs=30,
batch_size=batch_size, verbose=1)
```

После завершения обучения модели была выполнена оценка производительности модели на валидационном наборе: вывод данных потерь и точноности. Данные показаны в листинге 9.

Листинг 9 – Оценка производительности

```
# Оценка производительности модели на валидационном наборе данных
validation_generator = datagen.flow_from_directory(
    validation_data_dir,  # Путь к директории с валидационными данными
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False  # Не перемешивать данные
)
Loss, accuracy = model.evaluate(validation_generator, steps=len(valida-
tion_generator))
print("Потери на валидационных данных:", loss)
print("Точность на валидационных данных:", accuracy)
```

Визуализация процесса обучения позволяет показать путь активации нейронов от входных нейронов через активацию признаков и до активации выходного нейрона, показывающего метку класса. В процессе обучения строятся графики, отображающие изменение точности и потерь на тренировочных и валидационных данных. Это позволяет оценить эффективность модели и выявить проблемы. График точности (accuracy) показывает изменение точности модели на тренировочных данных в зависимости от эпох. График потерь (loss) показывает изменение потерь модели на тренировочных данных в зависимости от эпох. Аналогичные графики потель т точности на валидационных данных также представлены. Листинг 10 показывает, как строятся соответствующие графики.

Листинг 10 – Визуализация

```
plt.figure(figsize=(12, 6))
# График точности на тренировочных данных
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Точность (accuracy)')
plt.xlabel('Эпохи')
plt.ylabel('Точность')
plt.title('Точность на тренировочных данных')
plt.legend()

# График потерь на тренировочных данных
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Потери (loss)')
plt.xlabel('Эпохи')
plt.ylabel('Потери')
plt.title('Потери на тренировочных данных')
plt.legend()
plt.show()
```

Обученная модель была сохранена в файле с помощью метода `save`: `model.save('my_model18_(30)_test.h5')`. Файлы формата .h5 (HDF5) является стандартным форматом для сохранения моделей Keras. Они содержат все параметры модели, включая веса, архитектуру и конфигурацию оптимизатора. Поэтому после сохранения модели в формате .h5 можно загрузить ее в любом другом приложении, где есть поддержка Keras или TensorFlow, и использовать для предсказаний на новых данных.

Выбранная модель имеет несколько преимуществ, которые делают ее эффективным инструментом для решения задачи классификации изображений.

1. Используя сверточные слои CNN позволяют эффективное извлекать признаки из изображений и обнаруживая локальные шаблоны и структуры.

2. Архитектура модели Sequential API позволяет построить нейронную сеть последовательно, добавляя слои один за другим. Это делает процесс создания модели более интуитивно понятным и удобным.

3. Стандартные слои, такие как Conv2D, MaxPooling2D, Flatten и Dense, что делает ее легко адаптируемой и настраиваемой.

4. В модели можно использовать слои Dropout, которые помогают предотвратить переобучение путем случайного исключения нейронов во время обучения.

5. Оптимизатор adam является эффективным методом обучения нейронных сетей, который автоматически адаптирует скорость обучения на основе градиентов.

6. Функция потерь categorical_crossentropy хорошо подходит для задач многоклассовой классификации.

Выбранная модель обладает всеми этими преимуществами, что делает ее мощным и эффективным инструментом для решения задачи классификации изображений. Для использования данной модели в приложении предполагается использовать код, представленный в листинге 11.

Листинг 11 – Загрузка модели

```
from tensorflow.keras.models import load_model
# Загрузка модели из файла .h5
model = load_model('my_model16.04.h5')
# Пример использования модели для предсказания
result = model.predict(test_data)
```

3.4. Эксперименты на наборе данных

Для определения эффективности нейронной сети и ее способности решать задачу классификации изображений можно провести ряд экспериментов.

Перед созданием модели необходимо определиться с размерами изображений и мини пакета. Выбор данных параметров влияет на производительность нейронной сети. В таблице 1 показано влияние Batch_size (мини пакета) на точность обучения при прочих равных показателях.

Таблица 1 – Изменение Batch_size

Эпохи	Нейроны в слоях	Width height	Batch size	Dropout	Скорость обучения	L2	Точность
10	32, 64	128x128	16	0,5	0,001	Да	0,8695
10	32, 64	128x128	8	0,5	0,001	Да	0,8907

В таблице 2 можно увидеть, как от изменения размера изображения изменяется точность обучения. В данном примере видно переобучение нейронной сети при размере изображения 100x100 пикселей.

Таблица 2 – Изменение размера изображения.

Эпохи	Нейроны в слоях	Width height	Batch size	Dropout	Скорость обучения	Точность
20	32, 64	100x100	32	0,5	0,01	0,9909
20	32, 64	28x28	32	0,5	0,1	0,8827

Большое влияние на производительность нейронной сети оказывает архитектура модели. Точность работы сети, ее способность к обобщению зависит от количества сверточных и полносвязных слоев, нейронов в слоях, функций активации и оптимизаторов, наличии слоев, позволяющих избежать переобучение (например, Dropout) и других параметров. Рассмотрим некоторые из них.

Изменение количества слоев может повлиять на производительность сети: увеличить ее способность к обобщению, но может также привести к переобучению. Базовая модель нейронной сети показана на листинге 12.

Листинг 12 – Модель нейронной сети

```
model = Sequential([
    Conv2D(32, (3, 3), input_shape=(img_width, img_height, 3), activa-
tion='relu'),
    MaxPooling2D(pool_size=pool_size),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=pool_size),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(num_classes, activation='softmax')
])
```

Модель представляет собой сверточную нейронную сеть (CNN) с двумя сверточными слоями, двумя слоями пулинга и двумя полносвязными слоями. Попробуем добавить третий сверточный слой, не меняя другие показатели. Можно увидеть, что точность обучения для данной выборки с увеличением количества слоев уменьшается, при чем, с уменьшением скорости обучение данный показатель увеличивается. Результаты представлены в таблице 3.

Таблица 3 – Сравнение количества сверточных слоев

Эпохи	Нейроны в слоях	Width height	Batch size	Dropout	Скорость обучения	Точность
20	32, 64	28x28	32	–	–	0,9088
20	32, 64, 128	28x28	32	–	–	0,8943
20	32, 64	28x28	32	0,5	0,01	0,8827
20	32, 64, 128	28x28	32	0,5	0,01	0,7586

В качестве функции активации используется ReLU для всех слоев, кроме последнего, где используется softmax для получения вероятностей принадлежности к классам. Добавим L2-регуляризацию к ядрам свертки во всех слоях для уменьшения переобучения. Скорость выполнения обучения при этом увеличивается. Результаты показаны в таблице 4.

Таблица 4 – Применение L2

Эпохи	Нейроны в слоях	Width height	Batch size	Dropout	Скорость обучения	L2	Точность
7	32, 64, 128	28x28	32	0,5	–	0,01	0,0829
10	32, 64	28x28	16	0,5	Step	0,05	0,4848
10	32, 64	128x128	16	0,5	0,001	0,01	0,8695
10	32, 64	128x128	8	0,5	0,001	0,01	0,8907
10	32, 64	128x128	16	0,5	0,0001	0,05	0,8472
10	32, 64	256x256	32	0,5	0,0001	0,01	0,1196
10	32, 64, 128	28x28	32	–	–	0,01	0,7888
15	32, 64	28x28	16	0,25	0,001	0,05	0,6946
20	32, 64	28x28	16	0,5	0,001	0,01	0,9285
20	32, 64, 128	28x28	32	0,05	–	0,01	0,0865
20	32, 64, 128	28x28	32	–	–	0,01	0,8063
30	32, 64	28x28	32	0,25	0,01	0,01	0,8639
30	32, 64, 128	28x28	32	0,5	0,001	0,01	0,7870
30	32, 64, 128	28x28	32	–	–	0,01	0,8765

Далее было проанализировано влияние замедление скорости обучения (`learning_rate`), коэффициента L2, количества эпох и размер мини пакета на обучение. Значительное уменьшение точности показывает использование пошагового уменьшения с сочетанием слоя Dropout с коэффициентом 0,5 на 16 эпохах. Схожие результаты можно увидеть на 15 эпохах с коэффициентом 0,25 слоя Dropout. При этом вместо пошагового уменьшения скорости используется `learning_rate` = 0,001. Данные приведены в таблице 5.

Таблица 5 – Влияние различных параметров

Эпохи	Нейроны в слоях	Width height	Batch size	Dropout	Скорость обучения	L2	Точность
7	32, 64, 128	28x28	32	0,5	–	0,01	0,0829
10	32, 64	128x128	8	0,5	0,001	0,01	0,8907
10	32, 64	128x128	16	0,5	0,001	0,01	0,8695
10	32, 64	256x256	32	0,5	0,0001	0,01	0,1196
10	32, 64	128x128	16	0,5	0,0001	0,05	0,8472
10	32, 64	28x28	16	0,5	пошаговое	0,05	0,4848
15	32, 64	28x28	16	0,25	0,001	0,05	0,6946
20	32, 64	28x28	16	0,5	0,001	0,01	0,9285
20	32, 64, 128	28x28	32	0,05	–	0,01	0,0865
20	32, 64, 128	28x28	32	–	–	0,01	0,8063
30	32, 64	28x28	32	0,25	0,01	0,01	0,8639
30	32, 64, 128	28x28	32	0,5	0,001	0,01	0,7870

Из таблицы видно, что количество эпох не может быть меньше 10. Четко прослеживается зависимость точности обучения от размера изображения (с увеличением размера падает качество обучения) и скорости обучения: при добавлении параметров, понижающих скорость обучения желательно увеличивать количество эпох. Так, при размере изображения 256x256 пикселей точность обучения 0,1196. Как и говорилось ранее, необходимо подбирать количество слоев к соответствующим параметрам.

Помимо оптимизатора Adam регулировать скорость обучения можно и другими способами. Например, пошаговое уменьшение скорости обучения (Step Decay) позволяет начать с большой скорости и затем, каждые несколько эпох (или определенного количества шагов) уменьшать скорость на определенный коэффициент. Step Decay показан на листинге 13.

Листинг 13 – Step Decay

```
initial_learning_rate = 0.1
     lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
         initial_learning_rate, decay_steps=1000, decay_rate=0.96, stair-
case=True)
     optimizer = tf.keras.optimizers.SGD(learning_rate=lr_schedule)
```

Экспоненциальное уменьшение (Exponential Decay) позволяет более плавно уменьшать скорость обучения по мере приближения к локальному

оптимуму. В этом случае, скорость обучения снижается экспоненциально с каждой эпохой. Пример кода показан на листинге 14.

Листинг 14 – Exponential Decay

```
    initial_learning_rate = 0.1
    lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
        initial_learning_rate, decay_steps=1000, decay_rate=0.96, stair-
case=False)
    optimizer = tf.keras.optimizers.SGD(learning_rate=lr_schedule)
```

Примеры применения Step Decay и Exponential Decay показаны в таблице 6. Данные эксперименты показали, что применение поэтапного уменьшение скорости на изображениях 100x100 приводят к переобучению модели независимо от количества эпох и размера мини пакета. Применение L2 совместно с поэтапным уменьшением скорости не существенно влияет на результаты. Из приведенных данных видно, что точность при применении Step Decay или Exponential отличается мало. Дополнительное использование Dropout и L2, напротив, приводит к недоученности модели.

Таблица 6 – Step Decay и Exponential

Эпохи	**Нейроны в слоях**	**Width height**	**Batch size**	**Dropout**	**Скорость обучения**	**L2**	**Точность**
10	32, 64	28x28	16	–	Step	–	0,8973
10	32, 64	28x28	16	–	Exponential	–	0,8957
10	32, 64	100x100	16	–	Step	–	0,9964
10	32, 64	100x100	16	–	Exponential	–	0,9961
10	32, 64	100x100	32	–	Step	–	0,9961
10	32, 64	100x100	32	–	Exponential	–	0,9945
10	32, 64	128x128	16	–	Step	–	0,9728
10	32, 64	28x28	16	0,5	Step	–	0,4028
10	32, 64	28x28	16	0,5	Step	0,05	0,4848

В следующем блоке экспериментов для увеличения и разнообразия набора данных был создан генератор данных для обучения модели, который применяет различные преобразования к изображениям во время обучения. В данном примере рассматриваются следующие преобразования: вращение, сдвиг, масштабирование и отражение изображения. Каждое преобразование можно корректировать коэффициентом. Код генератора представлен на листинге 15.

Листинг 15 – Генератор изображений

```
train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True
```

В таблице 7 показаны эксперименты с использованием генерации изображений.

Таблица 7 – Применение генерации изображений

Эпохи	Нейроны в слоях	Width height	Batch size	Dropout	Скорость обучения	Точность
10	32, 64	128x128	16	0,001	0,001	0,7373
10	32, 64	256x256	16	0,5	0,01	0,4574
20	32, 64	28x28	32	0,5	0,01	0,8907
30	32, 64, 128	28x28	32	0,5	0,0001	0,5828
50	32, 64	28x28	32	0,05	–	0,5828

Обучение на 30 и 50 эпохах не дает положительного результата. Данная сеть показывает плохие результаты при большом размере изображения.Наиболее приемлемые результаты можно увидеть на 20 эпохах при размере изображение 28x28 с применением слоя Dropout и уменьшения скорости 0,01.

Часто возникает необходимость использовать предобученную нейронную сеть в создаваемой сети. Использование предобученной модели в проекте имеет ряд преимуществ, в частности, перенос обучения. При этом можно заморозить нижние слои модели, чтобы сохранить уже выученные признаки, и обучить только верхние слои под конкретную задачу. Это особенно полезно, когда есть ограниченное количество данных для обучения и нет возможности обучить модель с нуля. В данном проекте была использована модель сверточной нейронной сети VGG16. К сожалению, сеть VGG имеет два серьезных недостатка: очень медленная скорость обучения большой вес архитектуры. Код с использованием VGG представлен на листинге 16.

Листинг 16 – Использование модели VGG16

```
# Загрузка предварительно обученной модели VGG16 без полносвязных слоев
base_model = VGG16(weights='imagenet', include_top=False, in-
put_shape=(img_width, img_height, 3))
# Замораживаем веса базовой модели, чтобы не обучать их в процессе fine-
tuning
base_model.trainable = False
# Создаем сверточную модель поверх предварительно обученной модели
model = Sequential([
    base_model,
    Flatten(),
    Dense(256, activation='relu'),  # Регуляризация L2 убрали
    Dropout(0.5),  # Dropout для уменьшения переобучения
    Dense(num_classes, activation='softmax')
])
# Уменьшение скорости обучения
optimizer = Adam(learning_rate=0.01)  #
# Компиляция модели
model.compile(optimizer='adam', loss='categorical_crossentropy', met-
rics=['accuracy'])
# Предварительное обучение (fine-tuning)
# Размораживаем несколько последних слоев базовой модели
for layer in base_model.layers[-4:]:
    layer.trainable = True
```

Использование модели VGG16 показало очень низкую обучаемость сети для используемого набора данных. Сочетание предварительно обученной сети и L2 не рекомендуется применять для решение нашей задачи. Данный метод требует больших временных затрат: около 4 часов требовалось на выполнение кода. Рассматривались одинаковы параметры: количество нейронов в слое Dense 256, Width_height 32, Batch_size 32. Результаты экспериментов показаны в таблице 8.

Таблица 8 – Использование модели VGG16

Эпохи	**Dropout**	**Скорость обучения**	**L2**	**VGG**	**Точность**
15	0,25	0,001	0,01	VGG16 сохранение слоев	0,2132
30	0,5	0,0001	–	VGG16	0,5522
50	0,25	0,001	0,01	VGG16 сохранение слоев	0,2802
50	0,5	0,0001	–	VGG16 сохранение слоев генерация	0,7433

Итоги экспериментов представлены в таблицах, позволяя оценить влияние различных параметров на точность обучения. К сожалению, все

представленные выше оценки точности относились к тренировочным данным. Точность на валидационных данных не поднималась выше 50%. Это показывает, что нейронная сеть не справляется с поставленной задачей. Данные оценки точности на тренировочном и валидационном наборе данных представлены на рисунке 17.

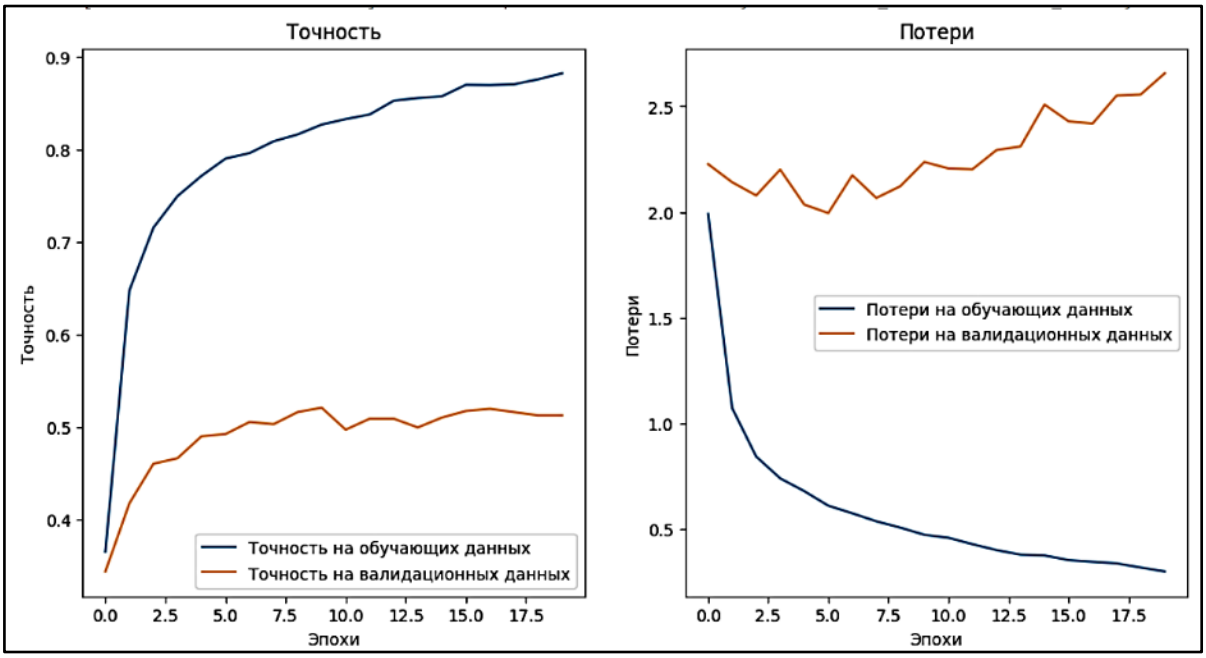

Рисунок 17 – Оценка точности и потерь

Исходя из вышесказанного, было принято решение, увеличить количество изображений используя метод аугментации: добавление новых слоев. Новые слои позволят отображать изображения по горизонтали, вращать изображения, отдалять или приближать, менять контраст:

1) `layers.experimental.preprocessing.RandomFlip("horizontal",input_shape=(img_height,img_width,3);`

2) `layers.experimental.preprocessing.RandomRotation(0.1);`

3) `layers.experimental.preprocessing.RandomZoom(0.1);`

4) `layers.experimental.preprocessing.RandomContrast(0.2).`

Как уже отмечалось выше, увеличение количества изображений методом генерации не приводило к хорошей обучаемости нейронной сети. Поэтому, вместе с аугментацией применили другую функцию потерь:

```
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits
=True).
```

На рисунке 18 показан график точности и потерь выполненный на 20 эпохах при размере width_height 128x128.

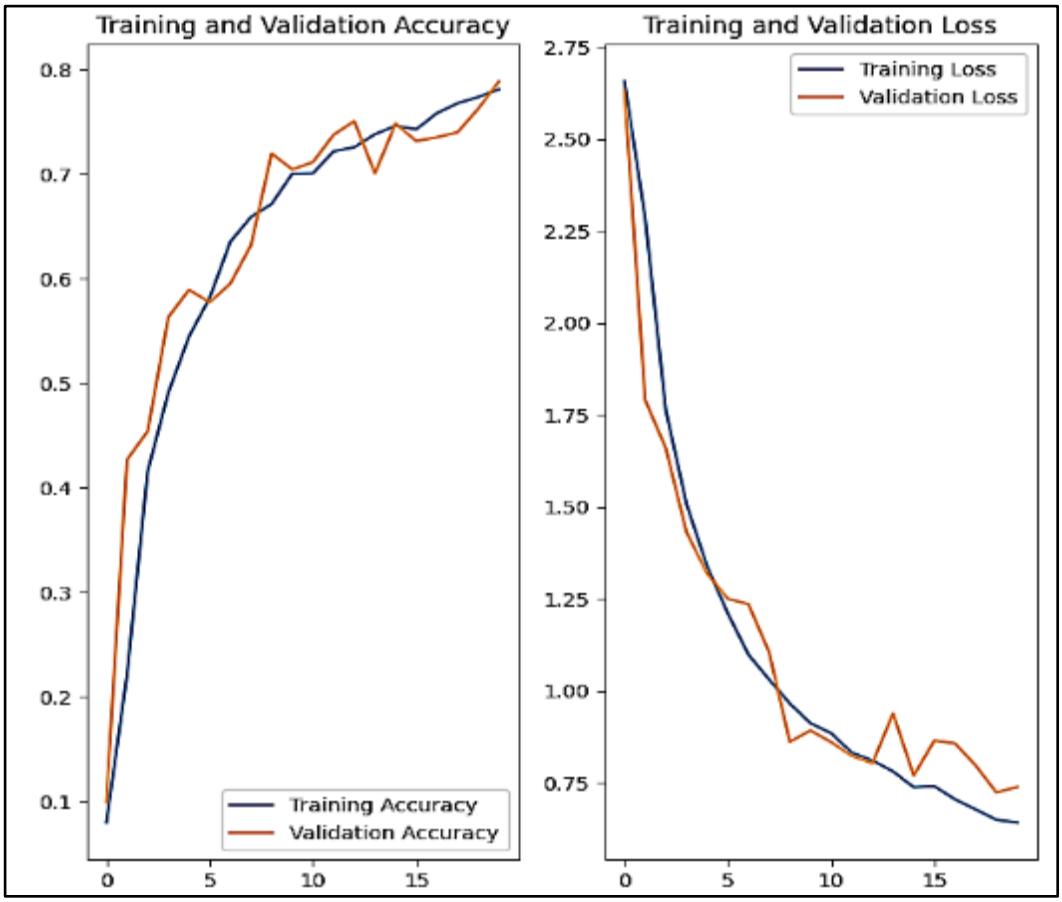

Рисунок 18 – Оценка точности и потерь

Далее, были поставлены эксперименты на: менялись размеры изображения: параметр width_height (от 28x28 до 256x256) и количество эпох. Экспериментально было установлено, что оптимальный результат получен на 30 эпохах и размере width_height 256x256. Результаты представлены в таблице 9.

Таблица 9 – Результаты экспериментов

n/n	Эпохи	Width height	Loss	Accuracy	Val loss	Val accuracy
1	20	128x128	0,6425	0,7813	0,7389	0,7888
2	30	128x128	0,5477	0,8169	0,6546	0,7995
3	30	28x28	1,001	0,6489	1,0863	0,6512
4	30	256x256	0,4555	0,8468	0,6319	0,8292

По результатам экспериментов остановились на варианте сверточной нейронной сети с использованием аугментации, функции потерь `loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)`, работающую на 30 эпохах с изображениями 256x256.

Рисунок 19 показывает, какие применены слои и количество нейронов в слоях.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 128, 128, 3)       0

 random_flip (RandomFlip)    (None, 128, 128, 3)       0

 random_rotation (RandomRot  (None, 128, 128, 3)       0
 ation)

 random_zoom (RandomZoom)    (None, 128, 128, 3)       0

 random_contrast (RandomCon  (None, 128, 128, 3)       0
 trast)

 conv2d (Conv2D)             (None, 128, 128, 16)      448

 max_pooling2d (MaxPooling2  (None, 64, 64, 16)        0
 D)

 conv2d_1 (Conv2D)           (None, 64, 64, 32)        4640

 max_pooling2d_1 (MaxPoolin  (None, 32, 32, 32)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 32, 32, 64)        18496

 max_pooling2d_2 (MaxPoolin  (None, 16, 16, 64)        0
 g2D)

 dropout (Dropout)           (None, 16, 16, 64)        0

 flatten (Flatten)           (None, 16384)             0

 dense (Dense)               (None, 128)               2097280

 dense_1 (Dense)             (None, 14)                1806

=================================================================
Total params: 2122670 (8.10 MB)
Trainable params: 2122670 (8.10 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Рисунок 19 – Слои нейронной сети

Полный код представлен в приложении А.

3.5. Альтернативная нейронная сеть

Рассмотрим альтернативу сверточным нейронным сетям для задачи классификации изображений: полносвязные нейронные сети. Полносвязные нейронные сети представляют собой классический вид нейронных сетей, где каждый нейрон в одном слое соединен с каждым нейроном в следующем слое.

Полносвязные нейронные сети обычно имеют большее количество параметров, поэтому они могут быть более подвержены переобучению, особенно на больших изображениях. Однако они могут быть хорошими вариантами для небольших наборов данных или задач с низкой сложностью.

В альтернативном коде используется полносвязная нейронная сеть (FCNN) или многослойный перцептрон. По-прежнему, используется библиотека TensorFlow. Данная библиотека позволяет нам использовать класс Sequential для создания модели нейронных сетей путем последовательного добавления слоев. В созданной полносвязной нейронной сети в качестве первого слоя используется Flatten для преобразования изображения в одномерный вектор. Затем добавляются два полносвязных слоя Dense с функцией активации ReLU. Выходной слой также является полносвязным с функцией активации softmax, чтобы получить вероятности принадлежности к различным классам. Класс ImageDataGenerator для генерации пакетов изображений и метод `flow_from_directory` позволяющий создавать генератор данных из папки с изображениями также использовались в сверточной нейронной сети.

В качестве показателя обучения также, как и в сверточной нейронной сети, использовался оптимизатор Adam (позволяющий эффективно обновлять веса модели), функция потерь categorical_crossentropy и метрика точности, accuracy, которая показывает долю правильных предсказаний модели.

Модель обучается на тренировочных данных с помощью метода `fit`, который предусматривает количество шагов обучения за эпоху и количество эпох обучения. Модель также оценивается на валидационных данных с помощью метода `evaluate`.

После обучения модели генерируются графики потерь и точности как для тренировочного, так и для валидационного наборов данных.

Полный код нейронной сети представлен в приложении Б.

Сравнивая результаты выполнения базовой сверточной нейронной сети и полносвязной нейронной сети, следует отметить, что полносвязная нейронная сеть требует больше времени на выполнения поставленной задачи. При этом, результаты обучения, при аналогичных входящих данных значительно хуже. Для сравнения были взяты одинаковые параметры: размеры изображений, размер мини-пакета, компиляция модели и количество эпох.

Полносвязная нейронная сеть показала результаты: loss 1,6312, accuracy 0,4587, время (в секундах) 2004 с. У сверточной нейронной сети loss 0,3196, accuracy 0,8834, время (в секундах) 1137 с.

Полносвязная нейронная сеть в отличие от сверточных нейронных сетей, не учитывает пространственную структуру входных данных: каждый нейрон каждого слоя связан со всеми нейронами предыдущего слоя. Входные данные перед передачей представляются в виде одномерного вектора.

Так как в полносвязной нейронной сети каждый нейрон каждого слоя связан со всеми нейронами предыдущего слоя, то из-за большого количества обучаемых параметров существует риск переобучения. CNN имеет меньше параметров, что позволяет модели изучать обобщающие признаки и более эффективно использовать вычислительные ресурсы.

Таким образом, хотя обе сети используются для решения задач классификации, сверточная сеть, благодаря своей способности изучать пространственные признаки в изображениях, оказалась более эффективной для решения нашей задачи, чем полносвязная сеть.

ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломной работы была поставлена задача разработки и обучения нейронной сети для классификации задач ОГЭ по способам их решения. Для достижения данной цели были исследованы данных о задачах ОГЭ и способы их решения. Были исследованы существующие методы классификации задач и применение нейронных сетей в образовании, актуальные исследования, посвященных использованию искусственного интеллекта в сфере образования.

Был подготовлен набор данных для обучения нейронной сети, включающий в себя изображение графиков функций, изучаемых в основной школе: линейная функция, парабола, гипербола. Данные были собраны из открытых источников и работ учащихся. Часть данных были сгенерированы с использованием языка программирования Python. Данные были классифицированы на классы (14 классов) и разбиты на обучающую и тестовую выборку.

В работе представлен обзор различных архитектур нейронных сетей, включая многослойные персептроны (MLP), сверточные нейронные сети (CNN), рекуррентные нейронные сети (RNN), глубокие нейронные сети (DNN) и специализированные архитектуры. Была выбрана сверточная нейронная сеть.

Также в работе показана важность программных средств для реализации нейронных сетей и описаны используемые в данном проекте инструменты и библиотеки. TensorFlow выбран в качестве основного инструмента благодаря его гибкости и высокой производительности. Основные библиотеки и модули включают TensorFlow, NumPy, Matplotlib и модуль os. Также были использованы модули и классы из вышеперечисленных библиотек.

В работе описывается процесс создания, обучения и оценки нейронной сети для классификации изображений. В рамках проекта используется метод глубокого обучения для распознавания объектов на изображениях с использованием набора данных, содержащего изображения различных

классов. Описываются преимущества выбранной архитектуры модели, включая эффективное извлечение признаков, удобство использования Sequential API, настраиваемость и адаптивность оптимизатора и функции потерь. Далее рассматриваются эксперименты, начиная с базовой модели. Результаты экспериментов сравниваются, приводятся в работе и делаются выводы о влиянии параметров на качество модели.

В работе рассматривается полносвязная нейронная сеть, дается ее описание, реализация с использованием TensorFlow, описывается обучение результаты сравниваются с CNN. Делается вывод об обоснованности сверточной нейронной сети.

Работа сопровождается приложениями, включающими коды нейронных сетей для дальнейшего изучения и использования.

Таким образом, выполнение дипломной работы позволило получить полезные результаты в области классификации задач ОГЭ и может быть полезно как для практического использования, так и для дальнейших исследований в данной области.

Данная работа также важна с точки зрения внедрения информационных технологий в обучающий процесс. Прослеживаются не только очевидные плюсы: использование нейронной сети при создании обучающего приложения, что окажет существенную помощь как учителю, так и ученикам при подготовке к экзамену, но уже на стадии написания работы был виден живой интерес учащихся к данной теме: ребята помогали в сборе данных, интересовались ходом написания работы, активно интересовались программированием, использованием ИИ в жизни. На основании данного проекта возможны разработки курсов дополнительного образования по программированию, рассчитанные на разные возрастные категории учащихся, что является актуальным в современном образовании.

ЛИТЕРАТУРА

1. Панова М.С. Обзор документов международных и региональных организаций по вопросам искусственного интеллекта в образовании. – М.: Центр искусственного интеллекта МГИМО, 2022. – 18 с.

2. Как подготовить данные для машинного обучения. [Электронный ресурс] URL: https://practicum.yandex.ru/blog/podgotovka-dannyh-k-analizu (дата обращения: 10.01.2024 г.).

3. Подготовка данных для алгоритмов машинного обучения. [Электронный ресурс] URL: http://blog.datalytica.ru/2018/04/blog-post.html (дата обращения: 10.01.2024 г.).

4. Курзаева Л. В., Овчинникова И. Г., Конькова, Д. С. Управление качеством профессионального образования на основе компетентностного подхода. – М.: ФЛИНТА, 2017. – 152 с.

5. Закон от 29.12.2012 г. № 273-ФЗ «Об образовании в Российской Федерации» [Электронный ресурс] URL: https://www.consultant.ru/edu/student/study/links// (дата обращения: 10.05.2024 г.).

6. Приказ Минпросвещения России (Министерства просвещения РФ) от 04 апреля 2023 г. №232/551/551 «Об утверждении Порядка проведения государственной итоговой аттестации по образовательным программам основного общего образования» [Электронный ресурс] URL: https://www.garant.ru/products/ipo/prime/doc/406770182/ (дата обращения: 10.05.2024 г.).

7. Паттерсон Д., Гибсон А. Глубокое обучение с точки зрения практика. – М: ДМК Пресс, 2018. – 418 с.

8. Антонио Д., Суджит П. Библиотека Keras – инструмент глубокого обучения. Реализация нейронных сетей с помощью библиотек Theano и TensorFlow. – М: ДМК Пресс, 2018. – 294 с.

9. Коэльо Л. П., Ричарт В. Построение систем машинного обучения на языке Python. 2-е изд. – М: ДМК Пресс, 2016. – 302 с.

10. Приказ Министерства просвещения РФ от 31 мая 2021 г. № 287 «Об утверждении федерального государственного образовательного стандарта основного общего образования» [Электронный ресурс] URL: https://www.garant.ru/products/ipo/prime/doc/401333920/ (дата обращения: 10.05.2024 г.).

11. Приказ Минобрнауки России от 17 декабря 2010 г. № 1897 «Об утверждении федерального государственного образовательного стандарта основного общего образования» [Электронный ресурс] URL: https://docs.edu.gov.ru/document/8f549a94f631319a9f7f5532748d09fa/ (дата обращения: 10.05.2024 г.).

12. Письмо Рособрнадзора от 17.04.2023 N 04-103 [Электронный ресурс] URL: https://legalacts.ru/doc/pismo-rosobrnadzora-ot-17042023-n-04-103-o-napravlenii-aktualizirovannykh/ (дата обращения: 10.05.2024 г.).

13. Matplotlib documentation. [Электронный ресурс] URL: https://matplotlib.org/stable/index.html (дата обращения: 20.03.2024 г.).

14. Шакирьянов Э. Д. Компьютерное зрение на Python. Первые шаги. – М: Лаборатория знаний, 2021. – 163 с.

15. Приказ Министерства просвещения Российской Федерации от 18.05.2023 г. № 370 «Об утверждении федеральной образовательной программы основного общего образования» [Электронный ресурс] URL: https://www.garant.ru/products/ipo/prime/doc/407288976/ (дата обращения: 10.05.2024 г.).

16. Маккинни У. Python и анализ данных. 2-ое изд. – М: ДМК Пресс, 2020. – 540 с.

17. NumPy documentation. [Электронный ресурс] URL: https://numpy.org/doc/stable/ (дата обращения: 01.03.2024 г.).

18. Гудфеллоу Я., Бенджио И., Курвиль А. Глубокое обучение. 2-е изд. – М: ДМК Пресс, 2018. – 652 с.

19. Селянкин В.В. Компьютерное зрение. Анализ и обработка изображений: учебник для вузов. – Санкт-Петербург: Лань, 2021. – 152с.

20. Ясницкий Л. Н. Введение в искусственный интеллект. 2-е изд. – М.: Академия, 2008. – 174 с.

21. Воронина В. В. Теория и практика машинного обучения. – Ульяновск: УлГТУ, 2017. – 290 с.

22. Федеральная служба по надзору в сфере образования и науки. ФГБНУ «Федеральный институт педагогических измерений». [Электронный ресурс] URL: https://fipi.ru/ (дата обращения: 05.01.2024 г.).

23. Образовательный портал для подготовки к экзвменам «СДАМ ГИА». [Электронный ресурс] URL: https://sdamgia.ru/ (дата обращения: 11.03.2024 г.).

24. Ященко И. В., Высоцкий И. Р., Коновалов Е. А.: ОГЭ-2024. Математика. Типовые экзаменационные варианты. 36 вариантов. – М: Национальное образование, 2023. – 224 с.

25. Хайкин С. Нейронные сети: полный курс / С. Хайкин. - 2-е изд., испр; пер. с англ. – М.: И.Д. Вильямс, 2006. – 1104 с.

26. Keras: the Python deep learning API. [Электронный ресурс] URL: https://keras.io (дата обращения: 10.05.2024 г.).

27. TensorFlow. [Электронный ресурс] URL: https://www.tensorflow.org (дата обращения: 10.05.2024 г.).

ПРИЛОЖЕНИЯ

Приложение А. Листинг сверточной нейронной сети.

В листинге 1 представлен код сверточой нейронной сети.

Листинг 1 – Сверточная нейронная сеть

```
import matplotlib.pyplot as plt
import numpy as np
import PIL
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
image_folder='/content/drive/MyDrive/DIPLOM/train'

batch_size = 32
img_width = 256
img_height = 256

train_ds = tf.keras.utils.image_dataset_from_directory(image_folder, vali-
dation_split=0.2, subset="training",seed=123,im-
age_size=(img_height,img_width),
batch_size=batch_size)

validation_ds = tf.keras.utils.image_dataset_from_directory(
image_folder,
validation_split=0.2,
subset="validation",
seed=123,
image_size=(img_height,img_width),
batch_size=batch_size)
class_names = train_ds.class_names

#модель
num_classes = len(class_names)
model = Sequential([
layers.experimental.preprocessing.Rescaling(1./255,in-
put_shape=(img_height,img_width,3)),
layers.experimental.preprocessing.RandomFlip("horizontal",in-
put_shape=(img_height,img_width,3)),
layers.experimental.preprocessing.RandomRotation(0.1),
layers.experimental.preprocessing.RandomZoom(0.1),
layers.experimental.preprocessing.RandomContrast(0.2),
layers.Conv2D(16,3,padding='same',activation='relu'),
layers.MaxPooling2D(),
layers.Conv2D(32,3,padding='same',activation='relu'),
layers.MaxPooling2D(),
layers.Conv2D(64,3,padding='same',activation='relu'),
layers.MaxPooling2D(),
layers.Dropout(0.2)
layers.Flatten(),
layers.Dense(128,activation='relu'),
layers.Dense(num_classes])

model.compile( optimizer='adam',# loss=tf.keras.losses.SparseCategorical-
Crossentropy(from_logits=True), metrics=['accuracy']
)
model.summary()

#обучение модели
epochs=30
```

```
history=model.fit( train_ds, validation_data=validation_ds, epochs=epochs
)
#Просмотр результатов
acc=history.history['accuracy'] val_acc=history.history['val_accuracy']
loss=history.history['loss'] val_loss=history.history['val_loss']

epochs_range = range(epochs)
plt.figure(figsize=(8,8))
plt.subplot(1,2,1)
plt.plot(epochs_range,acc,label='Training Accuracy')
plt.plot(epochs_range,val_acc,label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1,2,2)
plt.plot(epochs_range,loss,label='Training Loss')
plt.plot(epochs_range,val_loss,label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
#тест для проверки модели
model.save_weights('/content/drive/MyDrive/DIPLOM/model_weights_GPU.h5')
from re import VERBOSE
model.save_weights('/content/drive/MyDrive/model_weights_GPU.h5')
#оценка модели
loss,acc=model.evaluate(train_ds,verbose=2)
print("Restored model, accuracy: {:5.2f}%".format(100*acc))

check_path='/content/drive/MyDrive/DIPLOM/for_check/h_1.1.jpg'
img=tf.keras.utils.load_img(
check_path,target_size=(img_height,img_width)
)
img_array=tf.keras.utils.img_to_array(img)
img_array=tf.expand_dims(img_array,0)

predictions=model.predict(img_array)
score=tf.nn.softmax(predictions[0])
print("На изображении скорее всего {} ({:.2f})% вероятность".format(
class_names[np.argmax(score)],
100*np.max(score)
))
plt.imshow(img)
plt.axis('off')
plt.show()
```

Приложение Б. Листинг полносвязной нейронной сети.

В листинге 2 представлен код полносвязной нейронной сети.

Листинг 2 – Полносвязная нейронная сеть

```
import os
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense
from tensorflow.keras.preprocessing.image import ImageDataGenerator

data_dir = '/content/drive/MyDrive/DIPLOM/dataset'# Папка с изображениями
img_width, img_height = 28, 28 # Размеры изображений и размер мини-пакета
batch_size = 32
# Создание генератора данных для тренировочного набора
train_datagen = ImageDataGenerator(rescale=1. / 255)

# Создание генератора изображений для тренировочного набора
train_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=True
)
num_classes = len(train_generator.class_indices) # Количество классов
(папок)

# Создание модели полносвязной нейронной сети
model = Sequential([
    Flatten(input_shape=(img_width, img_height, 3)),# Преобразование изоб-
ражений в одномерный вектор
    Dense(128, activation='relu'),
    Dense(64, activation='relu'),
    Dense(num_classes, activation='softmax')
])
# Компиляция модели
model.compile(optimizer='adam', loss='categorical_crossentropy',met-
rics=['accuracy'])
# Обучение модели
history = model.fit(train_generator, steps_per_epoch=len(train_generator,
epochs=10)

test_data_dir = '/content/drive/MyDrive/DIPLOM/test' #Папка с тестовыми
данными

# Создание генератора данных для тестового набора
test_datagen = ImageDataGenerator(rescale=1. / 255)

# Создание генератора изображений для тестового набора
test_generator = test_datagen.flow_from_directory(
    test_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',  #Несколько классов
    shuffle=False  # Не перемешиваем, так как у нас есть классы
)
loss, accuracy = model.evaluate(test_generator, steps=len(test_generator))
print("Потери на тестовых данных:", loss)
print("Точность на тестовых данных:", accuracy)
```