РЕФЕРАТ

Пояснительная записка 104 с., 51 рисунок, 14 источников, 4 приложения

ИСКУСТВЕННЫЙ ИНТЕЛЛЕКТ, ИНТЕЛЛЕКТУАЛЬНЫЕ СИСТЕМЫ, СИСТЕМЫ РАСПОЗНАВАНИЯ ОБРАЗОВ, ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ, МАШИННОЕ ОБУЧЕНИЕ, КЛАССИФИКАЦИЯ, РАЗРАБОТКА СИСТЕМЫ РАСПОЗНАВАНИЯ РУКОПИСНЫХ ЦИФР

Объектом исследования являются системы распознавания образов на базе машинного обучения.

Предмет исследования – вопрос актуальности применения систем распознавания образов рукописных цифр для автоматизации различных процессов.

В ходе выполнения магистерской диссертации был проведён подробный обзор объекта исследований и выполнен анализ предметной части. Была установлена цель проектной части. Была создана и протестирована система распознавания образов рукописных цифр, которую можно применять для решения различных задач.

В результате исследования были изучены нюансы создания и применения систем, основанных на искусственных нейронных сетях, и систем распознавания образов, в частности. Также были сделаны выводы об актуальности и выгодности применения подобных систем.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 6

1 ТЕХНОЛОГИЯ СИСТЕМ РАСПОЗНАВАНИЯ ОБРАЗОВ НА БАЗЕ МАШИННОГО ОБУЧЕНИЯ 7

1.1 Возможности применения систем распознавания образов 8

1.2 Достоинства и недостатки технологии 11

1.3 Вариант системы распознавания образов на базе МО 12

1.3.1 Предварительная обработка данных 14

1.3.2 Анализ структуры и параметров ИНС 15

1.3.2.1 Нейроны 16

1.3.2.2 Вес пути 17

1.3.2.3 Предвзятость (смещение) 17

1.3.2.4 Функция активации нейрона 18

1.3.3 Обучение ИНС 20

1.3.3.1 Целевая функция 21

1.3.3.2 Градиентный спуск 22

1.3.3.3 Обратное распространение ошибки 26

2 РАЗРАБОТКА СИСТЕМЫ РАСПОЗНАВАНИЯ ОБРАЗОВ РУКОПИСНЫХ ЦИФР 32

2.1 Синтез системы распознавания образов 35

2.1.1 Алгоритм для создания моделей ИНС 36

2.1.2 Создание собственного набора данных 47

2.1.3 Алгоритм предобработки данных 53

2.1.4 Тестирование и модернизация модели 62

2.1.5 Алгоритм для эксплуатации модели 69

2.2 Сегментация изображений 72
ЗАКЛЮЧЕНИЕ 75
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 77
ПРИЛОЖЕНИЕ А 79
ПРИЛОЖЕНИЕ Б 83
ПРИЛОЖЕНИЕ В 85
ПРИЛОЖЕНИЕ Г 86

ВВЕДЕНИЕ

Современные информационные технологии и системы могут включать в себя огромное количество процедур и этапов, которые моделируют или поддерживают процессы анализа различных массивов данных. Одной из таких процедур является задача классификации данных по некоторым критериям, которые задаются пользователем.

Человеческий мозг поразителен в плане восприятия, обработки и классификации данных. Например, на рисунке 1 представлены написанные от руки цифры “3”. Удивительно то, что, просто посмотрев на эту строку по-разному написанных цифр, человеческий мозг способен классифицировать её как одну информационную единицу – цифру “3”. Так ЭВМ не сможет решить задачу классификации для цифр, написанных от руки, тривиальным программированием, так как для неё эти образы цифр являются несвязанным набором пикселей. Чтобы заставить ЭВМ распознавать подобные образы, прибегают к использованию искусственного интеллекта. Его использование в задачах классификации образов на примере распознавания рукописных цифр является основной темой данной дипломной работы.

Рисунок 1 – Написанные от руки цифры “3”

1 ТЕХНОЛОГИЯ СИСТЕМ РАСПОЗНАВАНИЯ ОБРАЗОВ НА БАЗЕ МАШИННОГО ОБУЧЕНИЯ

Теория распознавания образов — раздел информатики и смежных дисциплин, развивающий основы и методы классификации и идентификации предметов, явлений, процессов, сигналов, ситуаций и тому подобных объектов, которые характеризуются конечным набором некоторых свойств и признаков. Такие задачи решаются довольно часто, например, при переходе или проезде улицы по сигналам светофора. Распознавание цвета загоревшейся лампы светофора позволяет принять правильное решение о том, можно или нельзя переходить или проезжать перекрёсток [1].

Во втором десятилетии двадцать первого века интеллектуальные системы получили экспоненциальное развитие. Интеллектуальная система — это техническая или программная система, способная решать задачи, традиционно считающиеся творческими, принадлежащие конкретной предметной области, знания о которой хранятся в памяти такой системы.

Распространённый современный подход создания интеллектуальных систем подразумевает под собой использование искусственных нейронных сетей (ИНС) – сетей искусственных нейронов, которые подражают работе человеческого мозга и способны обучаться. Первой такой системой был персептрон Розенблатта. Персептрон Розенблатта – это нейронная сеть, которая представляет собой алгоритм для выполнения двоичной классификации. Он определяет, относится ли объект к определенной категории (например, является ли животное на рисунке кошкой или нет). Фрэнк Розенблатт, родоначальник персептрона. Персептрон впервые вошел в мир в качестве аппаратного обеспечения. Розенблатт, психолог, который учился, а затем и читал лекции в Корнельском университете, получил финансирование

от управления по морским исследованиям в США, чтобы сконструировать машину, которая могла бы обучаться [2].

Розенблатт разработал однослойный персептрон. Это значит, что его аппаратный алгоритм не включал в себя несколько уровней, которые позволяют нейронным сетям моделировать иерархию признаков. Это была небольшая нейронная сеть, которая мешала персептрону выполнять нелинейную классификацию, например, вычислять значение функции XOR («ИСКЛЮЧАЮЩЕЕ ИЛИ»).

В настоящий момент уже существуют многослойные ИНС, имеющие миллионы самонастраивающихся параметров. Такие системы можно научить решать почти любые задачи. Использование ИНС подразумевает применение машинного обучения.

Машинное обучение (МО) – класс методов искусственного интеллекта, характерной чертой которых является не прямое решение задачи, а обучение за счёт применения решений множества сходных задач. Такая технология часто применяется для решения задач классификации объектов, например, распознавание образов на изображении (картинке, фото, видео и т.п.) [1].

1.1 Возможности применения систем распознавания образов

Системы распознавания образов (СРО) играют значительную роль в современном мире, поскольку они могут автоматически обрабатывать и анализировать большие объемы данных, извлекая из них полезную информацию. Необходимость в СРО возникает в самых разных областях — от военного дела и систем безопасности до оцифровки аналоговых сигналов. Это перспективное направление во многих сферах.

Безопасность и видеонаблюдение. Системы видеонаблюдения используют СРО для распознавания лиц, номерных знаков, а также для анализа

поведения людей и транспортных средств. Это помогает обеспечивать безопасность и предотвращать преступления. С 2017 года в Москве тестируется система умного видеонаблюдения с целью идентификации преступников в местах массового скопления людей. К городской сети камер подключена технология от российской компании «NTechLab», которая помогает задерживать правонарушителей. В Китае подобная система видеонаблюдения способна распознавать не только лица, но и марки автомобилей и одежды на людях, что может быть впоследствии использовано маркетологами для своих исследований [3].

Медицина и здравоохранение. СРО могут быть использованы для анализа медицинских изображений, таких как рентген, МРТ и КТ. Это помогает врачам и специалистам по диагностике быстрее и точнее определить заболевания и нарушения.

Распознавание образов является распространенной практикой в медицине — во многих случаях компьютеры позволяют замечать вещи, которые могут пропустить даже опытные врачи. СРО в данном случае являются вспомогательным инструментом, помогающим подтвердить или опровергнуть гипотезу врача или дать повод для более глубоких исследований.

В России ведутся разработки программных комплексов для диагностики раковых образований на снимках КТ, МРТ и ПЭТ. Для этого нейросеть заранее обучают на тысячах размеченных снимков, после чего точность распознавания новых снимков возрастает до 95-97 %. Среди прочих, разработкой такой платформы занимается департамент информационных технологий Москвы, используя открытую библиотеку Google TensorFlow.

Созданная Google ИНС «Inception» анализирует микроскопическое исследование биопсии лимфатических узлов в поиске раковых клеток в молочных железах. Для человека это очень долгий и трудоемкий процесс, в ходе которого легко ошибиться или пропустить что-то важное, так как в некоторых случаях размер изображения составляет 100 000 х 100 000

пикселей. Нейросеть «Inception» обеспечивает чувствительность около 92% против 72% у врача. ИНС не упускает из виду все подозрительные участки снимков, хотя и допускаются ложные срабатывания, которые позже может отфильтровать сам врач [3].

Автономные транспортные средства. СРО используются в автономных автомобилях и беспилотных летательных аппаратах для распознавания объектов, определения их положения и движения, а также для принятия решений о маневрах и соблюдении правил движения. Распознавание объектов в автомобилях — это необходимая часть систем безопасности ADAS (Advanced driver-assistance systems). ADAS могут быть реализованы как сложными средствами, вроде радара и инфракрасных датчиков, так и при помощи монокулярной камеры [3].

Промышленность и производство. СРО применяются на производственных линиях для контроля качества, определения дефектов и сортировки продукции. Это позволяет улучшить эффективность процессов и снизить производственные издержки.

Образование и научные исследования. СРО могут использоваться для анализа данных и обнаружения закономерностей в различных областях науки, таких как биология, химия, астрономия и другие. Это позволяет исследователям быстрее делать открытия и улучшать процесс обучения благодаря способностям нейронных сетей обнаруживать закономерности.

Системы распознавания образов имеют огромный потенциал и многообразные возможности применения в самых разных областях жизни. С развитием технологий, таких как искусственный интеллект и машинное обучение, эти возможности будут продолжать расти и улучшать качество жизни.

1.2 Достоинства и недостатки технологии

Технология распознавания образов на базе МО обладает рядом достоинств перед другими методами. МО на базе ИНС способны быстро адаптироваться к изменяющейся среде при незначительном переобучении.

Нейронные сети обладают потенциальным сверхвысоким быстродействием за счет использования массового параллелизма обработки информации.

Программы МО не подвержены к шумам во входных данных. Они имеют возможность работать при наличии большого числа неинформативных, шумовых входных сигналов. Нет необходимости делать их предварительный отсев, нейронная сеть сама определит их малопригодность для решения задачи и отбросит их [4]. Итого, можно выделить следующие преимущества:

— высокая устойчивость к входным шумам;
— отказоустойчивость;
— быстродействие;
— высокая адаптация к различного рода изменениям;
— широкий спектр применения.

Что касается недостатков такого подхода, то к ним можно отнести необходимость сбора огромного количества данных. Чтобы обучить систему распознавания, требуется собрать большую базу данных примеров распознаваемого образа. Например, той же цифры “3” на рисунке 1. Чтобы определить эту написанную от руки цифру, ИНС необходимо “скормить” множество подобных промаркированных образов. Этот процесс сам по себе затратен по времени.

Также, в любом случае, существует вероятность ошибки классификации, потому что на выходе ИНС даёт вероятность того, что определяемый объект

относиться к тому или иному классу. Данные, которые используются для обучения должны быть безошибочными, чего крайне трудно добиться, учитывая необходимый их объём для корректной и точной работы. Однако ошибку можно свести к минимуму и точность системы может достигать 99 %, в зависимости от поставленной задачи.

Синтез усложняется выбором структуры ИНС для конкретной задачи. Для этого требуется опыт и знания программиста. Разные структуры хороши для разных задач, но если попалась беспрецедентная задача, то выбирать их придётся кропотливым ручным подбором, тестируя модели с разными алгоритмами для поиска метода с наивысшей точностью.

Можно выделить следующие недостатики:

— сложность получения данных;
— системы на базе МО подвержены ошибкам;
— сложность выбора нужной структуры ИНС для конкретной задачи;

1.3 Вариант системы распознавания образов на базе МО

Ниже приведён вариант создания системы распознавания образов (СРО), учитывающий использование ИНС и МО. Полный процесс создания является достаточно сложным и время затратным, однако в сети интернет и в некоторых библиотеках языков программирования уже имеются готовые шаблоны и методы для синтеза моделей ИНС. В этой работе рассмотрен полный процесс создания для лучшего понимания изучаемой темы.

Поставим задачу, которую будет решать нейронная сеть. Она будет следующей: распознавать образы написанных от руки цифр от 0 до 9 с высокой точностью.

Как было сказано раньше, для обучения ИИ нам нужен набор однотипных проиндексированных данных, и чем он больше, тем точнее будет работать обученная нейронная сеть. Это объясняется тем, что полученная модель является аппроксимированной по выходным значениям функцией, где выходными значениями является набор обучающих данных. Этим фактом и поразительны ИНС, потому что, при должном количестве обучающих данных, они могут быть аппроксимированными функциями почти чего угодно.

В случае поставленной задачи уже существует такая база данных (БД) – MNIST data-base (сокращение от «Modified National Institute of Standards and Technology»). База данных является стандартом, предложенным национальным институтом стандартов и технологий США с целью калибровки и сопоставления методов распознавания изображений с помощью машинного обучения на основе нейронных сетей [5], [6]. База данных MNIST содержит 70 000 проиндексированных изображений 28x28 пикселей рукописных цифр [7]. В наборе данных MNIST изображения рукописных цифр представлены на черном фоне с белыми цифрами. Иными словами, числовые пиксели, соответствующие рукописным цифрам, имеют более высокое значение, а пиксели фона, которые обычно составляют большую часть изображения, имеют нижнее значение (часто нулевое). Часть БД представлена на рисунке 2.

Рисунок 2 – Часть БД "MNIST"

1.3.1 Предварительная обработка данных

Предварительно необходимо разделить подготовленную БД на две части: набор для обучения и набор для тестирования. В первой части будет 60 000 изображений, во второй – 10 000. Обучающая часть состоит из экземпляров, на которых ИНС будет учиться классифицировать рукописные цифры. Тестовая часть состоит из аналогичных экземпляров, которые не используются в обучении. Тестовая часть служит для проверки точности обученной ИНС на примерах, которые она ещё «не видела».

Следующим шагом будет нормализация входных в ИНС данных. То есть необходимо нормализовать входные изображения. ИНС лучше всего работают с одномерными массивами данных, содержащими значения от 0 до 1 включительно. Это уменьшает битовый вес данных, что, вследствие, увеличивает скорость обработки и обучения. Так как в рассматриваемом случае изображения являются матрицами пикселей размером 28x28, где каждый пиксель может иметь значение от 0 (полностью чёрный пиксель) до 255 (полностью белый пиксель), то необходимо преобразовать каждое изображение в массив из 784 значений в пределах от 0 до 1.

Чтобы преобразовать значение серости каждого пикселя (то есть то, насколько он чёрный или белый) в промежуток от 0 до 1 достаточно просто поделить каждое значение серости каждого пикселя на 255. А для преобразования матрицы в одномерный массив можно написать простой алгоритм или воспользоваться уже готовыми методами, например «Flatten()», который имеется в библиотеке «Tensorflow». Этот метод преобразовывает двумерный массив в одномерный, в котором строки двумерного массива записываются последовательно.

1.3.2 Анализ структуры и параметров ИНС

Рассмотрим структуру ИНС. Её можно визуализировать как на рисунке 3. ИНС состоит из нейронов, которые образуют между собой слои. ИНС имеет входной слой, скрытые слои и выходной слой. Каждый нейрон одного слоя связан с каждым нейроном соседних. Нейроны внутри одного слоя не связаны друг с другом. Такая структура ИНС называется многослойной полносвязной.

Входной слой – это слой входных данных. В рассматриваемой нейронной сети каждый нейрон входного слоя считывает значение серости соответствующего ему пикселя входного изображения. Количество нейронов этого слоя для изображений разрешением 28×28 пикселей тогда будет 784.

Скрытые слои – это слои обработки данных. Этих слоёв может быть неограниченное количество. Здесь происходят все вычисления нейронной сети. До конца не ясно как конкретно происходит распознавание образов в этих слоях, но один из вариантов рассмотрения этих слоёв подразумевает, что здесь ИНС учится находить паттерны – закономерности – в сочетании близких к друг другу пикселей и строит иерархию признаков.

Выходной слой – результирующий. В рассматриваемой ИНС будет 10 нейронов выходного слоя, где каждый нейрон соответствует цифре от 0 до 9. Выходное значение с нейрона выходного слоя будет в промежутке от 0 до 1. Оно будет означать насколько вероятно, что данное на вход ИНС изображение соответствует той или иной цифре [8].

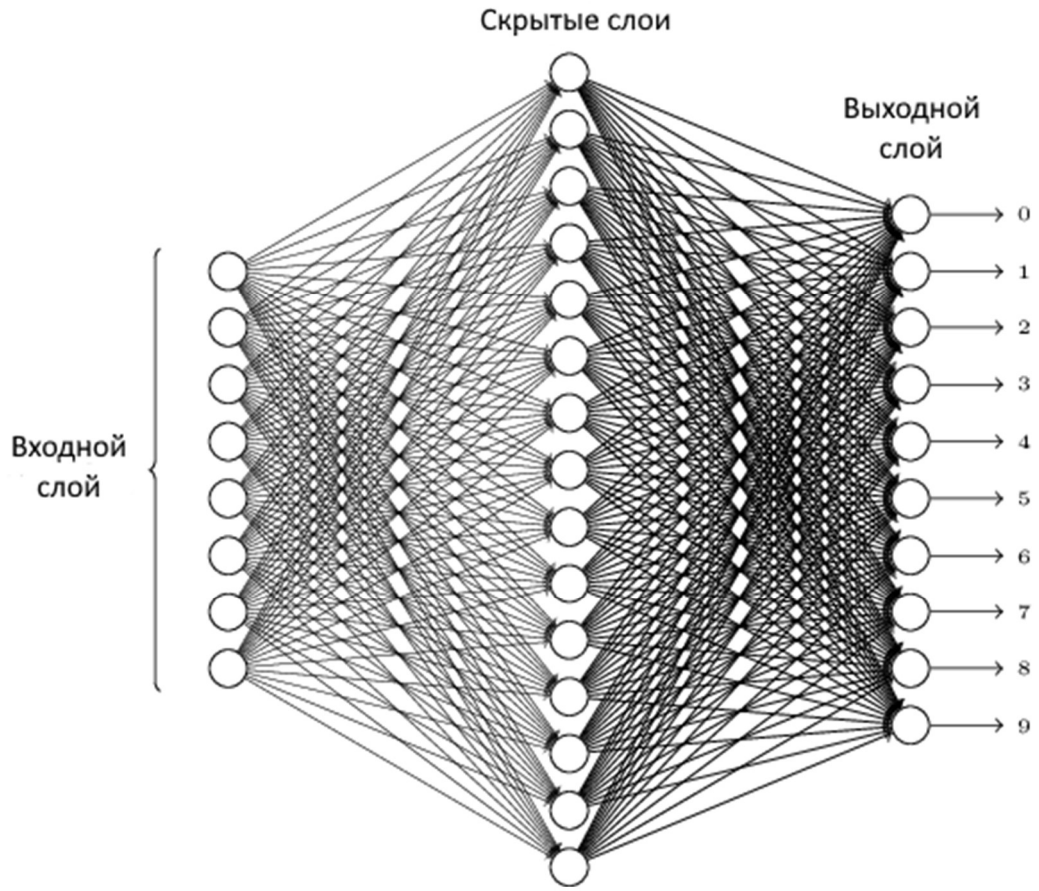

Рисунок 3 – Визуализация структуры ИНС

1.3.2.1 Нейроны

Рассмотрим нейрон как отдельную единицу и его функцию. Нейрон в ИНС – это математическая модель, которая имитирует работу биологического нейрона в мозге человека. Как и биологический аналог, искусственный нейрон принимает входные сигналы, обрабатывает их и производит выходной сигнал. В случае искусственных нейронов, выходной сигнал является линейной функцией. Такие нейроны объединяются в слои в ИНС, где каждый нейрон в слое связан с каждым нейроном в следующем слое. Входной слой принимает

входные данные, скрытые слои обрабатывают информацию и вычисляют выходные значения, а выходной слой представляет собой выходные значения ИНС. Каждый нейрон получает на вход нормализованную сумму обработанных некоторой связью значений каждого нейрона предыдущего слоя. Эта связь будет рассмотрена дальше. Как говорилось выше, нейрон входного слоя получает значение серости соответствующего ему пикселя изображения. На выходе нейрон выдаёт некое значение в промежутке [0;1] – это называется функцией активации нейрона, она обозначается как α. То есть при $0 < \alpha \leq 1$ – нейрон активирован, а при $\alpha = 0$ – не активирован. Условия активации нейрона будут рассмотрены дальше.

1.3.2.2 Вес пути

Выше было сказано, что каждый нейрон одного слоя связан с каждыми нейронами соседних слоёв. Основной элемент этой связи – вес пути. Обозначим его как w. Это первый настраиваемый за время обучения ИНС параметр. Он представляет из себя множитель, который обозначает связь двух соседних нейронов. Чем этот вес больше, тем сильнее связь между нейронами и тем больше вероятность активизации правого нейрона при активации связанного с ним левого нейрона. Почему это так, будет рассмотрено дальше.

1.3.2.3 Предвзятость (смещение)

Следующий численный настраиваемый ИНС параметр – предвзятость или, она же, смещение. Предвзятость представить как функцию барьера активации нейрона. То есть это значение, которое должна преодолеть сумма активаций нейронов предыдущего слоя, умноженных на соответствующие им веса. Предвзятость обозначается как b. Она служит для увеличения точности активации нейронов.

1.3.2.4 Функция активации нейрона

Условие активации нейрона следующее: нормализованная сумма произведений каждой активации нейронов предыдущего слоя на соответствующие им веса должна быть больше предвзятости правого нейрона. Математически это показано в формуле 1:

$$\sigma(w_1\alpha_1 + w_2\alpha_2 + \cdots + w_n\alpha_n - b) > 0, \quad (1)$$

где n – количество нейронов предыдущего слоя;

σ – сигмоида – функция активации, которая нормализует входные значения в промежуток от 0 до 1. Она представлена в формуле 2:

$$\sigma(\mathrm{Z}) = \frac{1}{1 + e^{-\mathrm{Z}}}, \quad (2)$$

где $\mathrm{Z} = \sum_n w_n\alpha_n - b.$

Тогда условие активации нейрона в случае сигмоиды можно записать как в формуле 3:

$$\begin{cases} \alpha = 1 \text{ при } Z \to \infty \\ \alpha = 0 \text{ при } Z \to -\infty \end{cases} \quad (3)$$

Графически сигмоида выглядит как на рисунке 4.

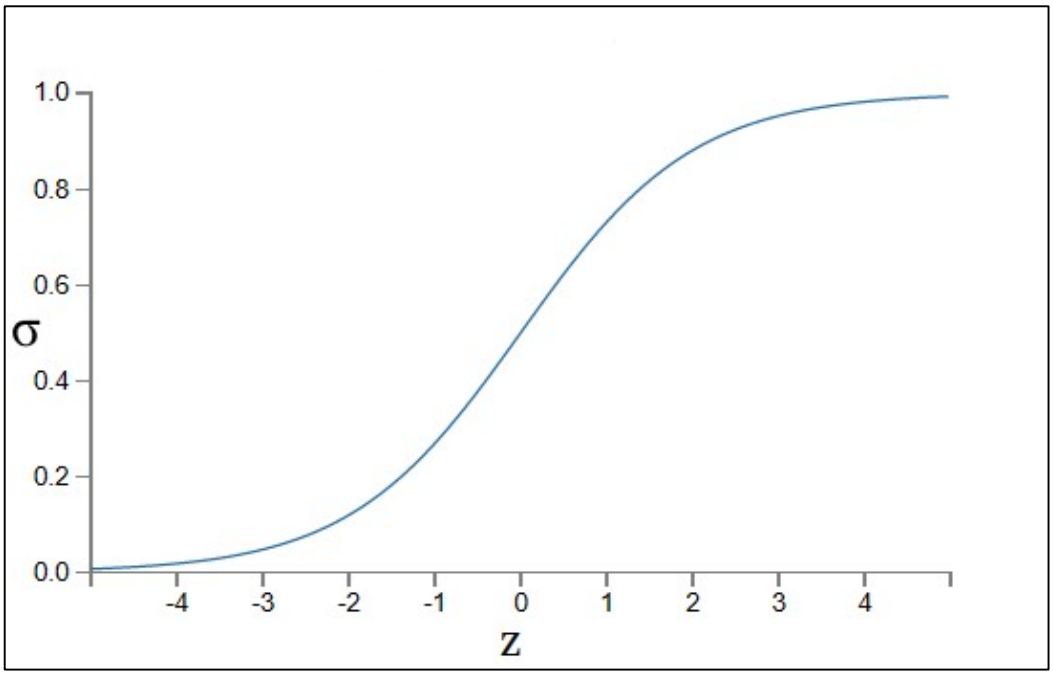

Рисунок 4 – Графическое представление функции сигмоиды

Как видно из рисунка 4, сигмоида является сглаженной версией шаговой функции, такой, как на рисунке 5. Если бы в ИНС действительно использовалась шаговая функция, то на выходе формулы 3 можно было бы получить только 0 или 1, в зависимости от того, больше ли Z нуля или меньше. Это бы был, так называемый, перцептрон. То, что сигмоида – это сглаженная шаговая функция, играет ключевой фактор самонастройки ИНС, так как при небольшом изменении параметров, входящих в Z (то есть w и b), выход с сигмоиды подвергается пропорционально небольшим изменениям и также наоборот: при большом изменении параметров – выход сильно меняется [8].

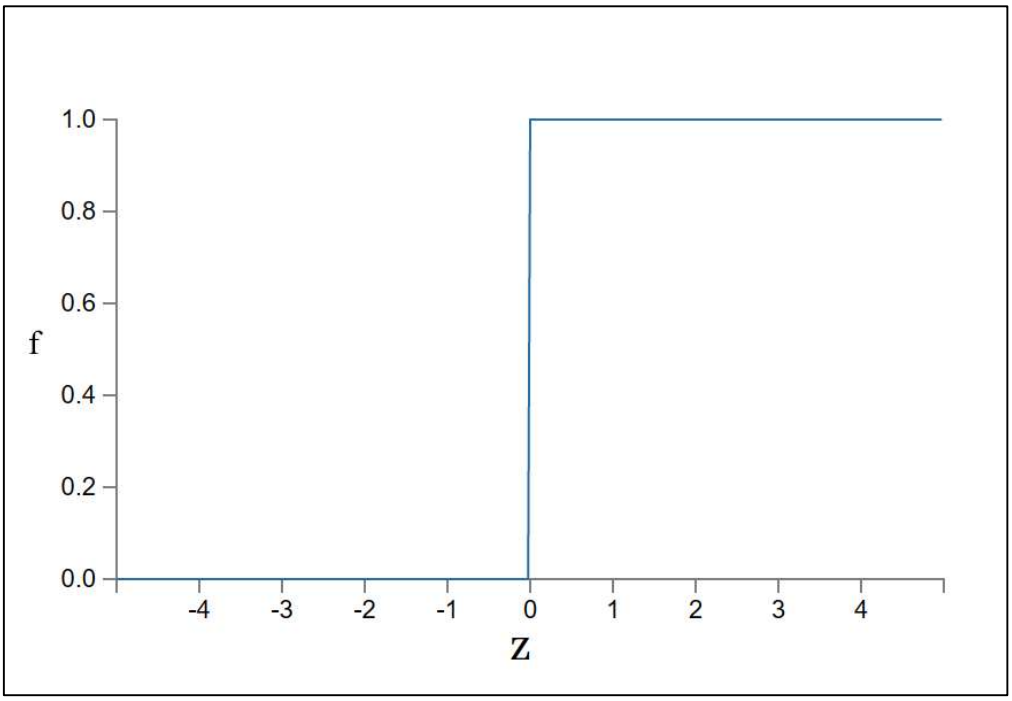

Рисунок 5 – Пример шаговой функции

1.3.3 Обучение ИНС

В данном подразделе рассмотрен принцип обучения ИНС за счёт самонастройки параметров с помощью обратного распространения ошибки (ОРО). В настоящий момент самый оптимальный вариант ОРО – это использование стохастического (или случайного) градиентного спуска (СГС). Градиентный спуск – это метод поиска глобального минимума функций. Стохастический градиентный спуск является оптимизированной версией градиентного спуска, потому что для поиска минимума он использует не весь набор данных, а только случайно выбранную часть анализируемых данных. Это отличие значительно ускоряет процесс обучения ИНС, потому что количество настраиваемых параметров могут исчисляться миллионами. Также СГС не допускает попаданий и зацикливаний в локальных минимумах в виду случайного выбора переменных функции. Принципы работы градиентного спуска будет рассмотрен ниже.

1.3.3.1 Целевая функция

Во время обучения через ИНС пропускают обучающий набор данных, чтобы обучить её решать задачу классификации на примерах решения схожих задач. Каждое изображения (или каждый элемент) набора проиндексировано соответствующей ей цифре. Основная задача ИНС – дать на выходе значение равное индексу входного изображения. Если проще, то на входное изображение с индексом 9 ИНС должна дать 9 на выходе. Обозначим входной элемент обучающего набора как x. Он представляет из себя $28 \times 28 = 784$-мерный вектор. Тогда индекс элемента x можно обозначить как $y(x)$, который является 10-мерным вектором в виде $y(x) = (0,0,0,0,0,0,0,0,0,1)^T$, где, в данном случае, он принадлежит одному из изображений цифры 9, так как 10-ый элемент соответствует именно этой цифре. Данный индекс является «желаемым выходным значением» для данного элемента x. «T» обозначает функцию транспонирования массива в вектор, так как градиент – это вектор, о чём подробнее будет сказано дальше.

Выходное значение ИНС зависит от входного элемента x и самонастраиваемых параметров w и b. В начале обучения параметры w и b принимают случайные значения. Требуется найти алгоритм, который позволяет найти все веса w и предвзятости b такие, что приближают выходные значения ИНС к желаемым выходным значениям $y(x)$ для всех входных элементов x. Чтобы численно определять эту близость вводится целевая функция $C(w, b)$ [8]. Уравнение для неё представлено в формуле (4):

$$C(w,b) \equiv \frac{1}{n}\sum_{i=1}^{n}\|y(x_i) - a(x_i)\|^2, \quad (4)$$

где: w – совокупность всех весов ИНС;

b – совокупность всех предвзятостей;

n – общее количество входных обучающих элементов;

$a(x_i)$ – выходной с ИНС 10-мерный вектор, соответствующий входному элементу x.

Из формулы 4 видно, что целевая функция – это среднеквадратичная ошибка выходного вектора ИНС от вектора желаемого выходного значения. Проанализировав формулу 4, можно прийти к следующим выводам: во-первых, функция $C(w,b)$ всегда положительна, так как все входящие переменные не отрицательны. Во-вторых, $C(w,b)$ становиться меньше при приближении выходных значений a к желаемым выходным значениям $y(x)$ для каждого элемента x. Это значит, что чем ближе целевая функция к нулю – $C(w,b) \approx 0$ – тем лучше работает алгоритм. Следовательно и наоборот: чем больше значение целевой функции, тем было меньше совпадений выходных значений с желаемыми выходными значениями для каждого элемента обучающего набора. Отсюда следует, что главной задачей настройки ИНС является минимизация целевой функции $C(w,b)$.

1.3.3.2 Градиентный спуск

Градиентный спуск и стохастический градиентный спуск (СГС) — это два оптимизационных алгоритма, используемые для обучения моделей машинного обучения, таких как нейронные сети. Они оба являются

итеративными методами для минимизации функции ошибок (целевой функции), основанными на вычислении градиента этой функции.

Градиентный спуск использует градиент функции ошибок для определения направления движения в каждой итерации. Градиент представляет собой вектор частных производных функции ошибок по каждому параметру модели. В каждой итерации параметры модели обновляются в направлении антиградиента с гиперпараметром, называемым «скорость обучения». Процесс повторяется до тех пор, пока изменение функции ошибок не станет достаточно малым или пока не будет достигнуто определенное количество итераций.

Основная идея градиентного спуска заключается в том, что функция ошибок будет убывать быстрее, если двигаться в направлении антиградиента. Однако градиентный спуск может быть медленным, поскольку на каждом шаге требуется вычисление градиента по всем обучающим данным.

Чтобы лучше понимать градиентный спуск, можно рассмотреть принцип поиска минимума для некой трёхмерной функции $C_3(v_1, v_2)$, которая графически представлена на рисунке 6, где v_1 и v_2 – переменные функции.

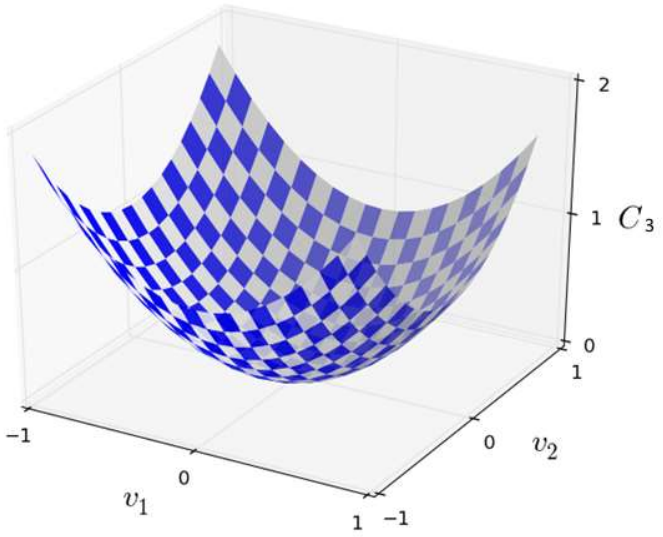

Рисунок 6 – Трёхмерная функция

Градиентный спуск ищет место, где функция достигает минимума. Сам по себе градиент является вектором, состоящим из частных производных функции по каждой переменной. Он показывает направление наибольшего возрастания функции в данной точке. Для функции $C_3(v_1, v_2)$ градиент в некой точке C расписывается как в формуле 5:

$$\nabla C_3 \equiv \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right). \tag{5}$$

Чтобы применить градиентный спуск и иметь возможность найти минимум функции необходимо провести конечное количество итераций с некоторым шагом в обратном градиенту направлении. Для этого можно ввести неотрицательный гиперпараметр η, который будет “коэффициентом” или “скоростью” шага. Он же является скоростью обучения. Тогда отдельный шаг функции в сторону уменьшения функции C_3 можно представить как в формуле 6:

$$\Delta C_3 = -\eta \left(\frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2\right), \tag{6}$$

где Δv_1 и Δv_2 – сдвиги в v_1 и v_2 в сторону уменьшения функции C_3 соответственно.

Если использовать матричную форму записи для формулы 6 и учитывать формулу 5, можно получить обобщённую запись шага для n-мерной функции в отрицательном направлении, которая записана в уравнение 7:

$$\Delta C_3 = -\eta \, \nabla C_3 \, \Delta v, \tag{7}$$

где, в данном случае:

$\Delta v = \begin{bmatrix} \Delta v_1 \\ \Delta v_2 \end{bmatrix}$ – матричный сдвиг переменных в сторону уменьшения функции $C_3(v_1, v_2)$.

Из уравнения 7 можно вывести сдвиг переменных Δv, который записан в формуле 8:

$$\Delta v = -\eta \nabla C_3 \tag{8}$$

Если обобщить, то градиентный спуск даёт возможность найти такие сдвиги переменных, которые ведут к наискорейшему уменьшению функции.

Стохастический градиентный спуск является вариацией градиентного спуска, которая призвана решить проблему медленных вычислений. Вместо использования всех обучающих данных для вычисления градиента на каждом шаге, СГС случайным образом выбирает один или несколько образцов из обучающих данных и использует их для вычисления приближенного градиента. Это делает каждую итерацию СГС быстрее, чем у градиентного спуска.

Алгоритм СГС добавляет стохастичность в процесс оптимизации, что может помочь избежать застревания в локальных минимумах и ускорить сходимость. Однако это также может привести к более шумным и менее стабильным процессам обучения, с множеством колебаний функции ошибок вокруг оптимального значения.

Между этими двумя алгоритмами существует компромисс между скоростью вычисления и стабильностью процесса обучения. Чтобы найти оптимальный баланс, часто используются вариации стохастического градиентного спуска, такие как мини-пакетный градиентный спуск (mini-batch gradient descent).

Вместо того чтобы использовать все обучающие данные или только один образец для вычисления градиента на каждом шаге, мини-пакетный градиентный спуск использует случайно выбранную группу образцов (мини-пакет). Это делает вычисления быстрее, чем в случае градиентного спуска и более стабильными, чем в случае стохастического градиентного спуска.

Мини-пакетный градиентный спуск сочетает преимущества обоих алгоритмов и является наиболее популярным вариантом оптимизации для обучения нейронных сетей. Размер мини-пакета (batch size) является важным гиперпараметром, который может влиять на производительность алгоритма и скорость сходимости.

В последние годы было предложено множество вариаций градиентного спуска и стохастического градиентного спуска, которые включают адаптивные методы изменения скорости обучения и момента. Примеры таких методов включают Adagrad, RMSprop, Adam, и другие. Они стремятся улучшить скорость сходимости и стабильность процесса обучения путем адаптивного изменения скорости обучения для каждого параметра модели в зависимости от истории градиентов.

В зависимости от задачи и данных, различные алгоритмы оптимизации и их вариации могут дать разные результаты. Поэтому важно экспериментировать с различными методами оптимизации и их гиперпараметрами для достижения наилучшей производительности обучаемой модели.

1.3.3.3 Обратное распространение ошибки

Обратное распространение ошибки (ОРО) является одним из наиболее популярных методов обучения ИНС. Он используется для определения оптимальных весов и смещений в ИНС, которые позволяют достигнуть наилучшей производительности сети на основе обучающих данных.

Процесс ОРО начинается с подачи входных данных на ИНС. Затем сигналы передаются по нейронам в прямом направлении до выходного слоя. В этом процессе каждый нейрон вычисляет свой выход, который зависит от входных данных и текущих весов.

После того, как выходные значения получены, происходит сравнение выходов с ожидаемыми значениями из обучающих данных. Ошибка, которая является разницей между выходом ИНС и ожидаемым выходом, рассчитывается для каждого нейрона в выходном слое (см. целевая функция).

Затем ошибка распространяется обратно через ИНС, начиная с выходного слоя и двигаясь к входному слою. На этом этапе вычисляются градиенты ошибок для каждого веса и смещения в ИНС.

Используя полученные градиенты, происходит обновление весов и смещений сети, чтобы минимизировать ошибку. Этот процесс повторяется множество раз, пока сеть не достигнет определенного уровня точности.

Математически этот процесс можно рассмотреть на примере двуслойной ИНС, где в каждом слое имеется по одному нейрону. Обозначим активацию правого нейрона как α^L, активацию левого как α^{L-1}. Тогда активацию правого нейрона a^L можно расписать как в формуле 9:

$$\alpha^L = \sigma(z^L), \tag{9}$$

где z^L расписывается по формуле 10:

$$z^L = w^L \alpha^{L-1} + b^L. \tag{10}$$

Допустив, что желаемое выходное значение с правого нейрона будет y (см. целевая функция), можно получить уравнение для целевой функции, которую обозначим как C_0. Это уравнение записано в формуле 11:

$$C_0 = (\alpha^L - y)^2. \quad (11)$$

Значение целевой функции C_0 зависит от параметра α^L, который в свою очередь зависит от параметра z^L, который зависит от w^L, α^{L-1}и b^L. Таким образом можно построить цепочку зависимостей. Это делается для того, чтобы понять, как любой отдельно взятый параметр ИНС влияет на её выходное значение и, следовательно, на целевую функцию. Это важный критерий обучения ИНС, потому что обучающему алгоритму необходимо знать какие параметры насколько сильно нужно сдвигать (настраивать) для наиболее эффективного уменьшения значения целевой функции. Например, чтобы определить, насколько влиятелен для C_0 параметр w^L, нужно рассчитать уравнение 12:

$$\frac{\partial C_0}{\partial w^L} = \frac{\partial z^L}{\partial w^L}\frac{\partial a^L}{\partial z^L}\frac{\partial C_0}{\partial \alpha^L} \quad (12)$$

где ∂ можно представить как очень маленький сдвиг к прикреплённому параметру.

Учитывая формулы (9), (10), (11), (12) получим формулу 13:

$$\frac{\partial C_0}{\partial w^L} = \alpha^{L-1}\sigma'(z^L)2(\alpha^L - y). \quad (13)$$

Таким образом была получена чувствительность целевой функции C_0 к изменениям веса w^L. Однако необходимо отметить, что эта чувствительность относиться только к одному конкретному экземпляру обучающего набора. Более точная чувствительность C_0 к w^L является средним арифметическим значением таких чувствительностей, полученных от всех обучающих

экземпляров, которые пропустили через ИНС. Тогда будет справедливо равенство, записанное в формуле 14:

$$\frac{\partial C_0}{\partial w^L} = \frac{1}{n} \sum_{i=0}^{n-1} \frac{\partial C_i}{\partial w^L}, \tag{14}$$

где n – количество экземпляров обучающего набора.

Чувствительность C_0 к w^L в формуле 14 является лишь одним параметром градиентного вектора ∇C_0. Чтобы получить полный вектор необходимо рассчитать чувствительность целевой функции C_0 к каждому параметру, входящему в ИНС.

Сейчас был рассмотрен пример для самой простой ИНС. Расчёты для более больших сетей несколько усложняются, но они остаются аналогичными. Рассмотрим вариант ОРО для двуслойной ИНС с произвольным количеством нейронов в каждом из слоёв. Для этого введём индексы: k для номера нейрона левого слоя и j для номера нейрона правого слоя. Тогда целевая функция C_0 будет записана как в формуле 15:

$$C_0 = \sum_{j=0}^{p^{(L)}-1} \left(\alpha_j^L - y_j\right)^2, \tag{15}$$

где $p^{(L)}$ – количество нейронов выходного (правого) слоя.

Вес между k-ым нейроном левого слоя и j-ым нейроном правого обозначим как w_{jk}^L. Тогда параметра z_j можно записать как в формуле 16:

$$z_j^L = \sum_{k=0}^{p^{(L-1)}-1} \alpha_k^{L-1} w_{jk}^L + b_j^L, \quad (16)$$

где $p^{(L-1)}$ – количество нейронов левого слоя. Здесь используется это значение, потому что оно равно количеству весов, привязанных к каждому нейрону правого слоя.

Тогда выражение из формулы 12 для данной ИНС запишется в виде формулы 17:

$$\frac{\partial C_0}{\partial w_{jk}^L} = \frac{\partial z_j^L}{\partial w_{jk}^L} \frac{\partial a_j^L}{\partial z_j^L} \frac{\partial C_0}{\partial \alpha_j^L}. \quad (17)$$

Как видно из выражения выше, суть вычислений для произвольного количества нейронов не поменялась и осталась аналогичной. Однако, стоит отметить одно различие. Оно заключается в исчислении чувствительности целевой функции C_0 к активации нейрона левого слоя α_k^{L-1}. Так как каждый нейрон левого слоя связан с каждым нейроном правого слоя, значение активации одного нейрона левого слоя влияет на значения активаций каждого нейрона правого. Поэтому в исчислении данной чувствительности используют сумму влияний этого нейрона. Выражение чувствительности целевой функции C_0 к активации нейрона левого слоя α_k^{L-1} записано в формуле 18:

$$\frac{\partial C_0}{\partial a_k^{L-1}} = \sum_{j=0}^{p^{(L)}-1} \frac{\partial z_j^L}{\partial a_k^{L-1}} \frac{\partial a_j^L}{\partial z_j^L} \frac{\partial C_0}{\partial \alpha_j^L}. \quad (18)$$

Стоит добавить, что для ИНС с произвольным количеством нейронных слоёв, по сути, ничего не меняется. Главное отличие – это бóльшая цепочка

зависимостей, которая была рассмотрена выше [9]. После того как были получены все значения параметров градиента ∇C_0, все веса и смещения сдвигаются в сторону уменьшения целевой функции C_0 так, как было описано в главе про градиентный спуск. Это и есть полный цикл одной итерации ОРО.

Важно отметить, что ОРО – это вычислительно затратный процесс, особенно при использовании больших ИНС. Однако благодаря различным техникам оптимизации, таким как стохастический градиентный спуск и использование графических процессоров (GPU), ОРО стал более доступным и практичным в использовании.

В целом, обратное распространение ошибки является одним из самых эффективных и широко используемых методов обучения ИНС. Он позволяет создавать более точные и адаптивные ИНС, которые могут решать самые разнообразные задачи, от классификации изображений до обработки естественного языка.

2 РАЗРАБОТКА СИСТЕМЫ РАСПОЗНАВАНИЯ ОБРАЗОВ РУКОПИСНЫХ ЦИФР

На основе изученного в первой главе материала можно приступить к созданию системы распознавания образов рукописных цифр на базе машинного обучения. Подобные системы уже имеют широкий спектр применения. Например, системы распознавания рукописных цифр используются для автоматической сортировки почтовых отправлений. Они распознают почтовые индексы и адреса, написанные вручную на конвертах или письмах, и помогают автоматизировать процесс сортировки и доставки [10].

В банках и финансовых учреждениях системы распознавания рукописных цифр используются для обработки чеков и других банковских документов. Они позволяют автоматически распознавать рукописные суммы, номера счетов и другую информацию, упрощая процесс обработки и уменьшая вероятность ошибок [11].

Некоторые приложения и устройства для перевода рукописного текста в печатный используют системы распознавания рукописных цифр. На мобильных устройствах системы распознавания рукописных цифр используются для ввода текста с помощью стилуса или пальца. Они позволяют пользователю писать рукописный текст на экране и автоматически распознавать его, преобразуя в печатный формат [12].

Также СРО можно использовать, например, для конвейерной сортировки товаров или сырья на предприятии или багажа на пассажирских логистических узлах. Применение технологии ИНС в этом случае может усовершенствовать принцип, по которому создают и наносят метки на товары, сырьё и багаж. В настоящий момент эти метки наносят в виде штрих- или QR-кодов на клейкую глянцевую бумагу, которую затем наносят на вышеперечисленные грузы.

Такой метод создаёт денежные затраты на бумагу и чернила для принтера, а также на печать и на наклейку меток идут временные затраты. Использование систем распознавания образов может сэкономить деньги и время, если создавать такие метки маркером от руки на контейнере груза. Также, в применении ИНС для решения задач сортировки, в зависимости от специфики и требований, можно выделить следующие преимущества:

— рукописные числа могут быть универсальными и не требуют специального оборудования для печати, в отличие от штрих- и QR-кодов;
— рукописные числа могут быть более устойчивыми к повреждениям или загрязнению по сравнению с штрих-кодами, которые могут стать нераспознаваемыми при небольшом повреждении;
— в некоторых случаях использование рукописных чисел может обеспечить большую гибкость, например, в ситуации, когда требуется быстро добавить дополнительную информацию на контейнер;
— для применения рукописных чисел не требуется специализированное оборудование для печати штрих-кодов.

Стоит помнить, что у использования рукописных цифр также есть свои недостатки, включая проблемы с непоследовательностью рукописного ввода, ошибками при вводе и возможностью трудностей с распознаванием определенных стилей письма, с чем нет проблем у штрих- и QR-кодов. Хоть эти проблемы делают применение СРО на базе МО для распознавания рукописных цифр на сегодняшний день не самым актуальным выбором в этой сфере, в будущем эти проблемы, скорее всего, будут решены. Для этого уже есть ряд подходов, которые можно будет реализовать в будущем:

— улучшенные модели обучения: по мере улучшения технологий глубокого обучения, модели становятся все более точными в распознавании рукописных символов, несмотря на их вариабельность;
— большие объемы данных: собирание большего количества данных для обучения может помочь моделям лучше понимать вариации в рукописном тексте;
— алгоритмы предобработки: Улучшенные алгоритмы предобработки могут помочь улучшить качество рукописных изображений перед их обработкой моделями распознавания;
— перенос обучения и многоязычное обучение: использование моделей, обученных на одном языке или наборе данных, для улучшения производительности на других языках или наборах данных также может помочь улучшить общую точность;
— улучшение аппаратного обеспечения: прогресс в области технологий сканирования и камер может также улучшить качество входных данных, увеличивая точность распознавания;
— ансамблевые методы: ансамблевые методы, которые комбинируют вывод из нескольких моделей, могут также улучшить общую производительность;
— обратная связь и итеративное обучение: модели могут быть обучены с использованием обратной связи от пользователей или экспертов, позволяя модели учиться на своих ошибках и постепенно улучшать свою производительность.

Целью проектной части диплома является создание и калибровка СРО рукописных цифр на базе МО, которую можно применять для различных целей и задач.

2.1 Синтез системы распознавания образов

В ИНС используется полносвязная многослойная последовательная структура. БД для обучения ИНС та же, что была описана в первой главе – MNIST Data Base. Выбранная среда разработки – Google Colab. Google Colab — сервис, созданный Google, который предоставляет возможность работать с кодом на языке Python через Jupyter Notebook, не устанавливая на свой компьютер дополнительных программ. В Google Colab можно применять различные библиотеки на Python, загружать и запускать файлы, анализировать данные и получать результаты в браузере. Google Colab является популярным выбором для создания и обучения ИНС, так как в нём уже есть нужные для синтеза библиотеки и методы, и в нём удобно работать с данными.

Помимо алгоритма создания моделей ИНС, будет необходим алгоритм предварительной обработки данных – изображений рукописных цифр – который будет форматировать изображения под тип изображений обучающего набор MNIST. Это необходимо, чтобы созданная ИНС могла работать с абсолютно новыми данными, связанными с эксплуатацией. Например, с фотографией с разрешением 800x800 пикселей цифры 7, написанной маркером на белом фоне на коробке, которую необходимо отсортировать на условный 7-ой путь. Чтобы ИНС могла обработать новое входное изображение, его нужно перевести в оттенки серого, инвертировать цвета и сжать до размеров 28x28 пикселей. Также изображения в БД MNIST отцентрованы по центру масс белых пикселей, что тоже нужно учитывать. Полученные изображения скорее всего будут иметь различные шумы, которые будут влиять на выходной результат модели в негативном ключе. Для решения или сглаживания этой проблемы есть несколько подходов, которые будут рассмотрены ниже.

Наконец, полезным для использования СРО будет небольшая программа, которая загружает модель, принимает на вход обработанное изображение,

нормализует его, пропускает его через загруженную модель и на выходе выдаёт класс объекта, то есть – цифру, написанную на изображении.

Таким образом СРО будет состоять из трёх алгоритмов: алгоритм для синтеза и обучения модели ИНС, алгоритм обработки входных данных и алгоритм использования модели.

2.1.1 Алгоритм для создания моделей ИНС

Создаёт и обучает модель будет алгоритм, написанный в Google Colab, который использует блочный подход к организации кода. Блок кода в Google Colab — это отдельный модуль, который можно запускать, редактировать и просматривать независимо от остальных блоков. Весь код представлен в приложении А.

Первый блок подключает библиотеки, которые позволяют создать и обучить модель. Фрагмент кода представлен на рисунке 7. Этот фрагмент кода импортирует несколько библиотек и модулей, которые широко используются в задачах машинного обучения и нейронных сетей. Их описание предоставлено на самом рисунке. Данный код готовит окружение для работы с TensorFlow и Keras, импортирует необходимые модули и определяет нужные функции и классы для создания и обучения нейронных сетей на примере задачи классификации рукописных цифр (MNIST).

Второй блок кода загружает изображения и их метки из БД MNIST в переменные. Второй блок представлен на рисунке 8, где mnist.load_data() во второй строке возвращает два кортежа, каждый из которых содержит два массива numpy. Эти два кортежа распаковываются в четыре отдельные переменные следующим образом: x_train и x_test представляют собой массивы numpy, содержащие обучающие и тестовые изображения соответственно; y_train и y_test — это массивы numpy, содержащие обучающие и тестовые метки соответственно.

Важно отметить, что эти переменные не являются просто числами или строками, они являются сложными структурами данных – массивами numpy, содержащими большое количество изображений и соответствующих меток.

Строка 4 присваивает названия классов для каждого изображения для удобства работы с данными.

```
1 import numpy as np
2 #используется для работы с массивами
3
4 import matplotlib.pyplot as plt
5 #библиотека для визуализации данных
6
7 %matplotlib inline
8 #команда, позваляющая выводить изображения прямо в блокнот
9
10 import tensorflow as tf
11 #фреймворк для работы и обучения ИНС
12
13 from tensorflow import keras
14 #API для создания и обучения нейронных сетей с использованием
  фреймворка Keras
15
16 from tensorflow.keras.datasets import mnist
17 #содержит наборы данных для практического использования в обучении
  моделей
18
19 from tensorflow.keras.models import Sequential
20 #представляет собой последовательную модель нейронной сети
21
22 from tensorflow.keras.layers import Dense
23 #Dense представляет собой полносвязный слой нейронной сети
```

Рисунок 7 – Фрагмент блока 1 с импортом библиотек

```
1 #загрузка изображений из БД MNIST
2 (x_train, y_train), (x_test, y_test) = mnist.load_data()
3 #добавление названий классов
4 class_names = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
```

Рисунок 8 – Блок 2 кода

Третий блок представлен на рисунке 9. Он нормализует данные в БД MNIST в значения от 0 до 1.

```
#изменение значений серого в пределы от 0 до 1
x_train = x_train / 255
x_test = x_test / 255
```

Рисунок 9 – Блок 3

Четвертый блок просто выводит часть БД MNIST с метками. Он представлен на рисунке 10. Вывод 4 блока представлен на рисунке 11.

```
#визуализация изображений из БД
plt.figure(figsize=(10,10))
for i in range (25):
plt.subplot(5,5, i +1)
plt.xticks([])
plt.yticks([])
plt.imshow(x_train[i], cmap=plt.cm.binary)
plt.xlabel(class_names[y_train[i]])
```

Рисунок 10 – Блок 4

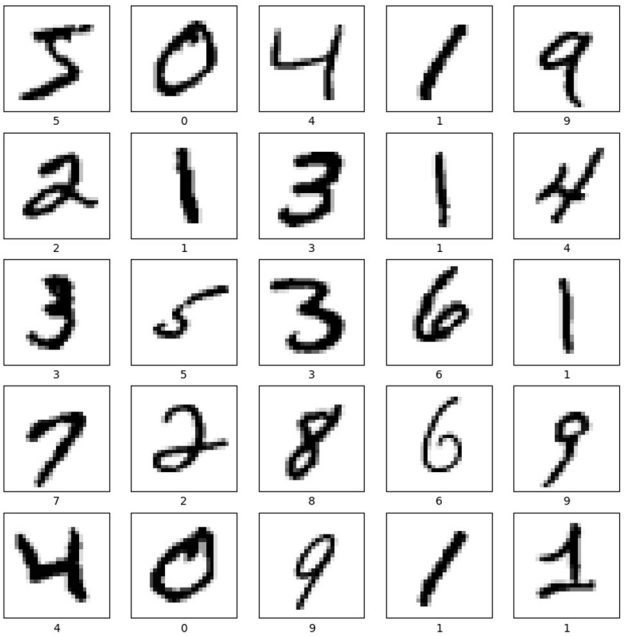

Рисунок 11 – Вывод с 4 блока

Пятый блок кода задаёт параметры последовательной ИНС, которую синтезирует метод библиотеки keras. Блок на рисунке 12.

```
1 #вызов метода создания ИНС с заданием кол-ва нейронов и слоев
2 model = keras.Sequential ([
3                             keras.layers.Flatten (input_shape=(28,28)),
4                             keras.layers.Dense (256, activation="relu"),
5                             keras.layers.Dense (128, activation="relu"),
6                             keras.layers.Dense (10, activation="softmax")
7                           ])
```

Рисунок 10 – Блок 5

Здесь строка 2 вызывает метод создания последовательной (sequential) ИНС.

Строка 3 задаёт входной слой. Метод layers.Flatten создаёт слой с заданным типом входных данных, где в данном случае, этот слой будет принимать массив $28 \times 28 = 784$ переменных.

Строки 4 и 5 задают скрытые полносвязные (dense) слои, где 256 и 128 – количество нейронов для второго и третьего слоя модели соответственно. Выбор количества нейронов в слоях для ИНС будет обоснован ниже. Команда activation = “relu” означает, что функцией активации для этих слоёв был выбран метод ReLU – Rectified Linear Unit (выпрямленная линейная единица). Эта функция выполняет ту же задачу, что и функция сигмоиды – нормализует выходные данные с некоторыми отличиями. Функция ReLU выражена в формуле 19:

$$ReLU(x) = \max(0, x), \tag{19}$$

где x – входная величина.

Функция $\max(0, x)$ возвращает любые значение x, если x больше или равно нулю и возвращает 0, если x меньше. Формулу 19 можно записать как формуле 20:

$$ReLU(x) = \begin{cases} x \text{ при } x \geq 0 \\ 0 \text{ при } x < 0 \end{cases}. \quad (20)$$

В отличии от функции сигмоиды (формула 2), которая возвращает значения в промежутке от 0 до 1, ReLU не ограничена сверху. Также ReLU возвращает 0 для отрицательных значений, в то время как сигмоида возвращает значения между 0 и 1 для любого входного значения. Сигмоида может страдать от проблемы исчезающего градиента, когда градиенты становятся очень маленькими для очень положительных или отрицательных входных значений. ReLU в некоторой степени решает эту проблему для положительных входных значений. Также ReLU значительно упрощает вычисления, по сравнению с другими функциями активации. Эта функция является наиболее часто используемой функцией активации в ИНС и в виду своих преимуществ, она была выбрана для скрытых слоёв в данной работе.

Строка 6 задаёт выходной слой с 10 нейронами, с функцией активации “softmax”. Функция softmax применяется в выходном слое нейронной сети для задач классификации для получения распределения вероятностей. Она преобразует набор чисел в вероятности, которые суммируются в 1. Функция softmax представлена в формуле 21:

$$softmax(z) = \frac{e^z}{\sum_{i=1}^{n} e^{z_i}}, \quad (21)$$

где z – входной вектор, в данном случае он включает все значения нейронов из выходного слоя;

z_i – элемент с номером i входного вектора;

n – количество элементов вектора, в случае ИНС – количество выходных нейронов.

Блок 6, на рисунке 11, вызывает метод обучения модели, который включает в себя обратное распространение ошибки и стохастический градиентный спуск.

```
1 #вызов метода обучения ИНС
2 model.compile(optimizer=tf.keras.optimizers.SGD(),
3               loss='sparse_categorical_crossentropy',
4               metrics=['accuracy'])
```

Рисунок 11 – Блок 6

В строке 2 блока 6 optimizer=tf.keras.optimizers.SGD() задает оптимизатор, который будет использоваться для обновления весов модели в ходе обучения. В этом случае используется стохастический градиентный спуск, а именно мини-пакетный СГС, где размер одного пакета задаётся в скобках. По умолчанию, если скобки пустые, размер пакета составляет 32 экземпляра. После вызова метода “model.compile” модель будет готова к обучению с помощью метода “.fit()”

Строка 3 определяет целевую функцию, которую модель будет стараться минимизировать в ходе обучения. Параметр “sparse_categorical_crossentropy” – это особый вид кросс-энтропийных потерь (значений целевой функции), который подходит для случаев, когда целевые метки представляют собой целые числа (а не векторы вероятностей).

Строка 4 задаёт параметр метрик, которые будут вычисляться в ходе обучения и тестирования. Они не влияют на процесс обучения, но позволяют отслеживать прогресс модели. В данном случае вычисляется точность классификации (accuracy).

Блок 7 представлен на рисунке 12. Он выводит на экран количество параметров ИНС и её структуру. Вывод с этого блока для данной ИНС представлен на рисунке 13.

```
1  #визуализация параметров ИНС
2  model.summary()
```

Рисунок 12 – Блок 7

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 256)               200960

 dense_1 (Dense)             (None, 128)               32896

 dense_2 (Dense)             (None, 10)                1290

=================================================================
Total params: 235,146
Trainable params: 235,146
Non-trainable params: 0
_________________________________________________________________
```

Рисунок 13 – Вывод с блока 7

Блок 8 запускает цикл обучения модели по эпохам, представлен на рисунке 14. Здесь в строке 2 вызывается метод “.fit()”, который обучает модель. На вход метода идут переменные с массивами изображений и метками, а также параметр эпох, который задаёт количество итераций обучения. К выбору количества эпох надо подходить аккуратно, чтобы не перенасытить её параметрами, которые отлично классифицируют обучающий набор, но не могут с хорошей точностью классифицировать тестовые данные. Этот параметр надо подбирать, основываясь на тестах разных моделей. К тому же нельзя исключать фактор случайности обучения ИНС в виду применения СГС, так как разные модели с одинаковыми параметрами могут вести себя по-разному.

```
1  #вызов метода обучения ИНС по эпохам
2  model.fit(x_train, y_train, epochs=15)
```

Рисунок 14 – Блок 8

На выходе блока 8, представленном на рисунке 15, показан процесс обучения ИНС в реальном времени, Там видны такие параметры как номер эпохи, время прохода одной эпохи, процент точности классификации тренировочных данных относительно их меток, значение целевой функции “loss” и т.д.

```
Epoch 1/15
1875/1875 [==============================] - 11s 5ms/step - loss: 0.5820 - accuracy: 0.8487
Epoch 2/15
1875/1875 [==============================] - 11s 6ms/step - loss: 0.2753 - accuracy: 0.9215
Epoch 3/15
1875/1875 [==============================] - 10s 5ms/step - loss: 0.2253 - accuracy: 0.9359
Epoch 4/15
1875/1875 [==============================] - 9s 5ms/step - loss: 0.1926 - accuracy: 0.9453
Epoch 5/15
1875/1875 [==============================] - 10s 6ms/step - loss: 0.1684 - accuracy: 0.9518
Epoch 6/15
1875/1875 [==============================] - 10s 5ms/step - loss: 0.1497 - accuracy: 0.9577
Epoch 7/15
1875/1875 [==============================] - 9s 5ms/step - loss: 0.1345 - accuracy: 0.9620
Epoch 8/15
1875/1875 [==============================] - 11s 6ms/step - loss: 0.1215 - accuracy: 0.9658
Epoch 9/15
1875/1875 [==============================] - 13s 7ms/step - loss: 0.1111 - accuracy: 0.9691
Epoch 10/15
1875/1875 [==============================] - 9s 5ms/step - loss: 0.1018 - accuracy: 0.9716
Epoch 11/15
1875/1875 [==============================] - 11s 6ms/step - loss: 0.0935 - accuracy: 0.9737
Epoch 12/15
1875/1875 [==============================] - 11s 6ms/step - loss: 0.0868 - accuracy: 0.9759
Epoch 13/15
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0803 - accuracy: 0.9774
Epoch 14/15
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0749 - accuracy: 0.9792
Epoch 15/15
1875/1875 [==============================] - 10s 6ms/step - loss: 0.0699 - accuracy: 0.9801
<keras.callbacks.History at 0x7fd585675e40>
```

Рисунок 15 – Вывод с блока 8

Следующий блок кода – блок 9. Он представлен на рисунке 16. Его задача вычислить и вывести точность модели относительно тестового набора, который не использовался в обучении ИНС. Вывод с блока на рисунке 17.

```
1  #выводит значение точности модели в отношении к тестовому набору
2  test_loss, test_acc = model.evaluate(x_test, y_test)
3  print('Точность теста: ', test_acc)
```

Рисунок 16 – Блок 9

```
313/313 [==============================] - 1s 4ms/step - loss: 0.0883 - accuracy: 0.9742
Точность теста:  0.9742000102996826
```

Рисунок 17 – Вывод с блока 9

Как видно по рисунку 17, точность модели относительно тестового набора составила 97.42%. Для дальнейших тестов следует сохранить модель. Для этого надо подключить соответствующую библиотеку, так что вернёмся к первому блоку. Заодно подключим библиотеки и модули, которые будут полезны и необходимы для качественной обработки самодельных тестовых данных, которая будет рассмотрена дальше. Они представлены с описанием на рисунке 18 фрагмента первого блока. Более подробное описание библиотек представлено ниже.

```
24
25  ######################################
26  ### библиотеки для работы с данными ###
27  ######################################
28
29  import cv2
30  #библиотека алгоритмов компьютерного зрения и обработки изображений.
31
32  from PIL import Image
33  #основной класс для работы с изображениями
34
35  from PIL import ImageOps
36  #модуль, позволяющий конвертацию изображения в градации серого
37
38  from PIL import ImageEnhance
39  # модуль, содержащий классы для улучшения изображений
40
41  from google.colab import files
42  #модуль, предоставляющий методы для загрузки и скачивания файлов
43
44  from google.colab import drive
45  #модуль, предоставляющий методы для подключения к Google Drive
46
47  import glob
48  #позволяет считывать названия файлов в директориях по шаблонам
49
50  import os
51  #позволяет выполнять различные операции, такие как управление файлами и директориями
```

Рисунок 18 – Второй фрагмент блока 1 с импортом библиотек для работы с данными

На рисунке 18 Строка 29 – импортирует OpenCV (Open Source Computer Vision Library) — это библиотека алгоритмов компьютерного зрения и обработки изображений. Она включает в себя несколько сотен алгоритмов компьютерного зрения, которые помогают в извлечении информации из изображений и видео.

Строки 32, 35, 38 – PIL (Python Imaging Library) / Pillow: Pillow — это форк библиотеки PIL (Python Imaging Library). PIL/Pillow поддерживает множество форматов изображений и предоставляет мощные возможности обработки изображений.

Строка 32 – import Image: основной класс для работы с изображениями.

Строка 35 – import ImageOps: модуль, содержащий некоторые базовые операции над изображениями, такие как конвертация в градации серого, обрезка, вращение и изменение размера.

Строка 38 – import ImageEnhance: модуль, содержащий классы для улучшения изображений. Предоставляет методы для регулировки яркости, контраста, четкости и цветового баланса изображения.

Строки 41, 44 – google.colab: Это модуль, специфичный для Google Colab.

Строка 41 – import files: модуль, предоставляющий методы для загрузки и скачивания файлов.

Строка 44 – import drive: модуль, предоставляющий методы для подключения к вашему Google Drive.

Строка 47 – import glob: модуль в стандартной библиотеке Python, который предоставляет функции для создания списков файлов из поиска по шаблонам в строках символов. Он может быть использован для чтения всех файлов в директории, которые соответствуют определенному шаблону.

Строка 50 – import os: модуль os в Python предоставляет функции для работы с операционной системой. Он позволяет выполнять различные

операции, такие как управление файлами и директориями, выполнение команд в командной строке, получение информации о системе и многое другое.

Десятый блок подключает Google Drive к среде программирования и сохранят модель в указанной директории. Представлен на рисунке 19.

```
#подключает облачный диск Google Drive
drive.mount('/content/gdrive')

#Сохраняет модель на облачном диске
model.save('/content/gdrive/MyDrive/Colab Notebooks/my_model')
```

Рисунок 19 – Блок 10

Таким образом была создана первая модель распознавания образов. Она была обучена на готовом открытом наборе данных MNIST БД.

2.1.2 Создание собственного набора данных

Полезным дополнением будет указать каким образом можно составлять подобные БД. Для этой цели, а также для тестирования собственных изображений и их обработки, в данном блокноте Google Colab был дополнительно написан код, состоящий из 4 блоков, позволяющий составлять БД с предварительной обработкой и проверять точность обученной ИНС на этом новом наборе, состоящем, в данном случае, из 10 изображений от 0 до 9.

Для первого шага создания своего набора было написано 10 цифр от 0 до 9 чёрной шариковой ручкой. Этот список был сфотографирован, и он представлен на рисунке 20.

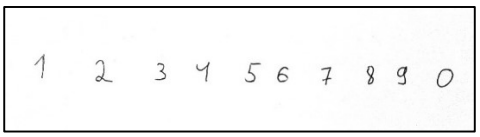

Рисунок 20 – Собственный рукописный список цифр

Следующим шагом было создание отдельных изображений каждой цифры. Для этого можно взять исходное изображение, вырезать и сохранить по отдельности каждую цифру в соотношении сторон 1 к 1. Это делается для соответствия обучающим данным модели, так как форматирование из другого соотношения сторон приведёт к искажению изображения, что усложнит работу для ИНС. В итоге, на примере цифры 5, получится изображение как на рисунке 21. В данном случае изображение имеет соотношение сторон 256x256 пикселей. Также оно размытое, нечёткое, местами шумное, имеет неправильную инверсию цветов и не отцентрованное по соответствующему алгоритму. В общем, изображение не соответствует обучающему набору MNIST.

Рисунок 21 – Рукописная цифра 5

Чтобы изображение соответствовало входным данным, необходимо преобразовать его в соответствующий формат. Изображение должно:

- быть в оттенках серого;
- иметь соотношение сторон 28x28 пикселей;
- быть максимально чётким, и с минимальным количеством шумов;
- содержать чёрный фон и белые цифры;
- быть отцентровано по центру масс белых пикселей.

С предварительной обработкой данных и созданием собственного тестового набора данных справляется блок 11 кода, который представлен на рисунках 22-23.

Фрагмент блока на рисунке 22 по большей части обрабатывает входные изображения в соответствии с требованиями, указанными выше. Однако подробный разбор с примерами работы каждой команды будет изложен дальше. Здесь будет акцент на создание набора данных.

В строке 4 создаётся вектор, в который будут сохраняться массивы пикселей каждого обработанного изображения, чтобы потом можно было их прогнать через созданную модель.

В строке 6 создаётся массив меток для соответствующих изображений, что требуется для проверки точности работы модели на новых данных.

В строке 9 инициализируется цикл, который каждый оборот берёт файл типа .png из указанной директории, где хранятся изображения для нового набора.

```
#пример создания собственного набора тестовых данных

#в этой матрице будут храниться массивы изображений
image_list = []
#в этом векторе будут храниться метки для каждого изображения
label_list = []

#цикл обработки и добавления в набор каждого изображения типа .png в обозначенной директории
for filename in glob.glob('/content/drive/MyDrive/Colab Notebooks/custom_dataset/Centr_Img/*.png'):

    #добавление изображения в переменную im
    im = Image.open(filename)
    #конвертация изображения в оттенки серого
    im = im.convert('L')
    #увеличение контраста для снижения шумов
    enhancer = ImageEnhance.Contrast(im)
    im = enhancer.enhance(2)
    #сохранение изображения и конвертация в массив
    im.save ('/content/123.png')
    im = cv2.imread('/content/123.png', 0)
    #добавление медианного размытия для снижения шумов
    im = cv2.medianBlur(im,1)
    #сохранение изображения из массива
    im = Image.fromarray(im)
    #повторное увелечение контраста
    enhancer = ImageEnhance.Contrast(im)
    im = enhancer.enhance(8.2)
    #изменение изображений в нужное разрешение с применением сглаживания
    im = im.resize((28, 28), Image.ANTIALIAS)
    enhancer = ImageEnhance.Contrast(im)
    im = enhancer.enhance(2.6)
    #инвертирует цвета (белый в чёрный и наоборот)
    im = ImageOps.invert(im)
    #нормализация
    im = np.array(im) / 255.0

```

Рисунок 22 – Первый фрагмент блока 11

Остаток фрагмента этого блока делает обработку изображений, поэтому рассмотрим второй фрагмент 11 блока на рисунке 23.

```
    # получает имя файла
    base = os.path.basename(filename)
    # разделяет имя файла на имя и расширение и преобразуем имя в целое число
    label = int(os.path.splitext(base)[0])
    label_list.append(label)
    image_list.append(im)


X = np.array(image_list)
Y = np.array(label_list)
```

Рисунок 23 – Второй фрагмент блока 11

Здесь строка 38 получает имя файла и сохраняет его в переменную base, чтобы потом использовать его для меток. Такой способ работает, если изображения были заранее проиндексированы в своём названии.

Строки 40-42 описаны в самом коде, а строки 45 и 46 сохраняют изображения и метки в соответствующие переменные так, как было описано выше.

Блок 12 аналогичен блоку 4 и просто выводит изображения полученного набора данных. Блок представлен на рисунке 24, вывод на рисунке 25.

```
1 #визуализация изображений из собственного набора
2 plt.figure(figsize=(10,10))
3 for i in range (10):
4   plt.subplot(5,5, i +1)
5   plt.xticks([])
6   plt.yticks([])
7   plt.imshow(X[i], cmap=plt.cm.binary)
8   plt.xlabel(class_names[Y[i]])
```

Рисунок 24 – блок 12

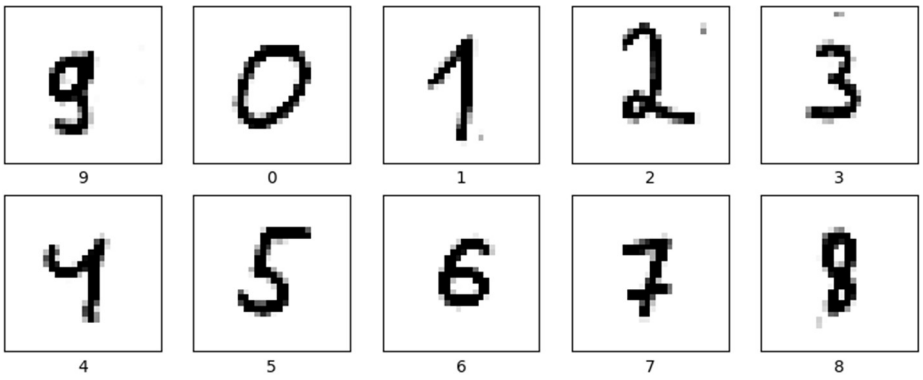

Рисунок 25 – вывод с блока 12

Из рисунка 25 видно, что на изображениях до сих пор остались шумы. Они, естественно, будут влиять на результаты тестирования этого набора. Однако, модель можно обучить так, чтобы она учитывала подобные искажения

на изображениях и умела их игнорировать. Как именно это делается, будет изложено ниже.

Блок 13 аналогичен блоку 9. Он выводит результаты тестирования в блокнот Google Colab. Блок и вывод представлены на рисунке 26.

```
test_loss, test_acc = model.evaluate(X, Y)
print('Точность теста: ', test_acc)
```

```
1/1 [==============================] - 0s 33ms/step - loss: 1.8217 - accuracy: 0.6000
Точность теста:  0.6000000238418579
```

Рисунок 26 – Блок 13 и его вывод

Из рисунка 26 видно, что точность определения составила 60%. Это означает, что только 6 из 10 рукописных цифр были определены правильно. Какие именно это были цифры можно узнать из блока 14. Он печатает номер изображения из матрицы тестовых данных, составленной в блоке 11, метку класса для этого изображения, предсказание модели для этого изображение и вероятность предсказанного класса в процентах. Блок представлен на рисунке 27, вывод блока на рисунке 28.

```
# Предсказание вероятностей классов для тестового набора
predictions = model.predict(X)

# Печать вероятностей
for i, prediction in enumerate(predictions):
    print(f'Изображение {i+1}, метка класса: {Y[i]}')
    print("Предсказание класса:", np.argmax(prediction))  # выведет предсказанную цифру
    print("С веротностью: {}%".format(round(np.max(prediction)*100, 2)))
    print('--------------------------------')
```

Рисунок 27 – Блок 14

```
1/1 [==============================] - 0s 24ms/step
Изображение 1, метка класса: 9
Предсказание класса: 1
С вероятностью: 58.55%
--------------------------------
Изображение 2, метка класса: 0
Предсказание класса: 0
С вероятностью: 82.46%
--------------------------------
Изображение 3, метка класса: 1
Предсказание класса: 1
С вероятностью: 85.95%
--------------------------------
Изображение 4, метка класса: 2
Предсказание класса: 2
С вероятностью: 98.66%
--------------------------------
Изображение 5, метка класса: 3
Предсказание класса: 5
С вероятностью: 51.61%
--------------------------------
Изображение 6, метка класса: 4
Предсказание класса: 4
С вероятностью: 96.9%
--------------------------------
Изображение 7, метка класса: 5
Предсказание класса: 5
С вероятностью: 88.47%
--------------------------------
Изображение 8, метка класса: 6
Предсказание класса: 4
С вероятностью: 56.74%
--------------------------------
Изображение 9, метка класса: 7
Предсказание класса: 7
С вероятностью: 47.3%
--------------------------------
Изображение 10, метка класса: 8
Предсказание класса: 1
С вероятностью: 42.44%
--------------------------------
```

Рисунок 28 – Вывод блока 14

2.1.3 Алгоритм предобработки данных

Пока что можно проигнорировать точность модели на самодельном наборе и проанализировать алгоритм обработки данных. Это будет отдельный блокнот Google Colab и, соответственно, нумерация блоков кода в этом

подразделе будет относиться к нему и идти начиная с первого с припиской “АПД” (“Алгоритма Предобработки Данных”). Целиком код алгоритма предобработки данных находится в приложении Б. Первый блок АПД подключает библиотеки, которые уже были рассмотрены выше. Он представлен на рисунке 29.

```
import cv2
import numpy as np
from google.colab import drive
from google.colab import files
from PIL import Image
from PIL import ImageOps
from PIL import ImageEnhance
import matplotlib.pyplot as plt
%matplotlib inline

#позволяет просматривать изображения cv2 в блокноте
from google.colab.patches import cv2_imshow
```

Рисунок 29 – Блок 1 АПД

Блок 2 подключает облачный сервис Google Drive, аналогично блоку 10 в предыдущей главе. Представлен на рисунке 30. Стоит отметить, то что он полностью “закомментирован”. Это сделано для удобства рабочего процесса, потому что диск Google Drive достаточно подключить один раз и не имеет смысла обращаться к этой команде при каждой компиляции.

```
#подключает облачный диск Google Drive
#drive.mount('/content/gdrive')
```

Рисунок 30 – Блок 2 АПД

Рассмотрим работу алгоритма на примере цифры 0 из списка самодельного набора. Исходное изображение представлено на рисунке 31.

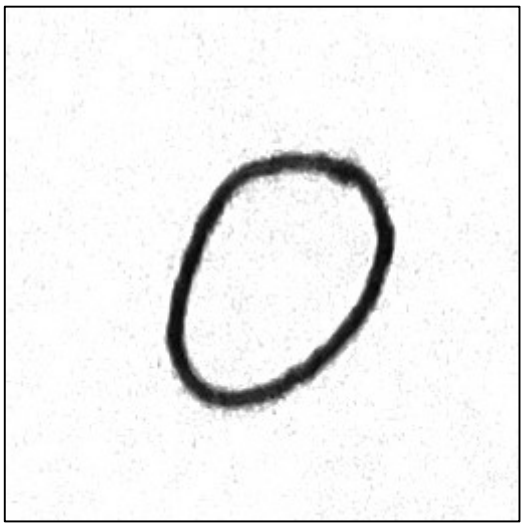

Рисунок 31 – исходное изображение

Это изображение было выбрано так как оно самое “шумное”, что наглядно может показать работу алгоритма предобработки данных. Итак, блок 3 АПД с его выводом представлены на рисунке 32.

Этот блок сохраняет исходное изображение в переменную img из указанной директории, переводит его в оттенки серого и увеличивает контраст в 2 раза для дальнейшей обработки. Дальше полученное изображение сохраняется во временное хранилище с названием 123.png для того, чтобы открыть его как массив numpy в следующем блоке. Это необходимо, чтобы применить медианное размытие, которое сильно глушит шумы изображения. И в конце блока изображение выводится на экран в блокноте.

```
#загрузка изображения в переменную img
img = Image.open('/content/drive/MyDrive/Colab Notebooks/custom_dataset/0.png')
#конвертация изображения в оттенки серого
img = img.convert('L')
#увеличение контраста
enhancer = ImageEnhance.Contrast(img)
img = enhancer.enhance(2)
#сохранение изображения для последующего открытия в OpenCV
img.save('123.png')
#вывод изображения в блокнот
plt.imshow(img)
plt.show()
```

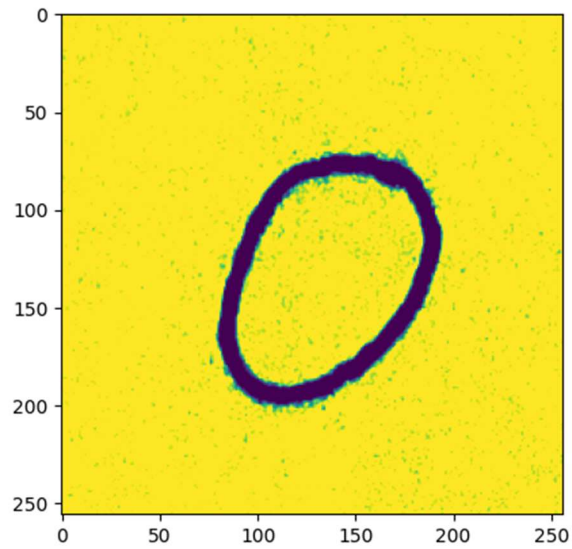

Рисунок 32 – Блок 3 АПД с его выводом

Блок 4 АПД применяет медианное размытие. Сам блок и его вывод на рисунке 33. Как видно из рисунка, изображение стало намного “чище”.

Строка 3 блока применяет метод медианного размытия, где на вход метода идёт массив изображения img и размер ядра, который используется для операции медианного фильтра. В данном случае, значение 5 означает, что будет использоваться квадратное ядро размером 5x5 пикселей.

```
#открытие изображения в OpenCV и добавление медианного размытия для снижения шумов
img = cv2.imread('/content/123.png', 0)
img = cv2.medianBlur(img,5)
cv2_imshow(img)
```

Рисунок 33 – Блок 4 АПД с его выводом

В контексте медианного фильтра, каждый пиксель изображения заменяется медианным значением пикселей в своей окрестности, которая определяется размером ядра. Например, для ядра размером 5x5, окрестность каждого пикселя будет состоять из 24 других пикселей вокруг него (и самого пикселя), и эти 25 значений будут отсортированы, а затем среднее (медианное) значение будет выбрано в качестве нового значения пикселя.

Блок 5 АПД сохраняет массив numpy обратно в виде изображения в переменную img, чтобы можно было вновь обрабатывать его с помощью модулей библиотеки PLT. Блок и его вывод представлены на рисунке 34.

```
#обратное сохранение изображения для обработки с помощью PLT
img = Image.fromarray(img)
plt.imshow(img)
plt.show()
```

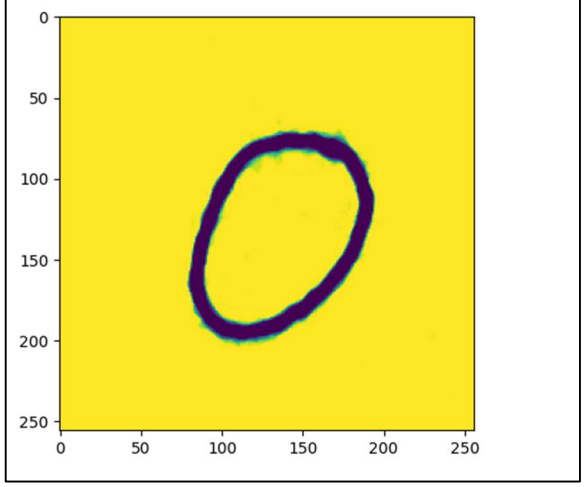

Рисунок 34 – Блок 5 АПД с его выводом

Блок 6 АПД увеличивает контраст в 8.2 раза для увеличения чёткости изображения. Это значение, которое оказалось наилучшим для данного набора. Затем изображение вновь сохраняется во временное хранилище 123.png для дальнейшей отцентровки с помощью модулей OpenCV. Блок и его вывод на рисунке 35.

```
#увеличение контраста
enhancer = ImageEnhance.Contrast(img)
img = enhancer.enhance(8.2)
img.save('123.png')
plt.imshow(img)
plt.show()
```

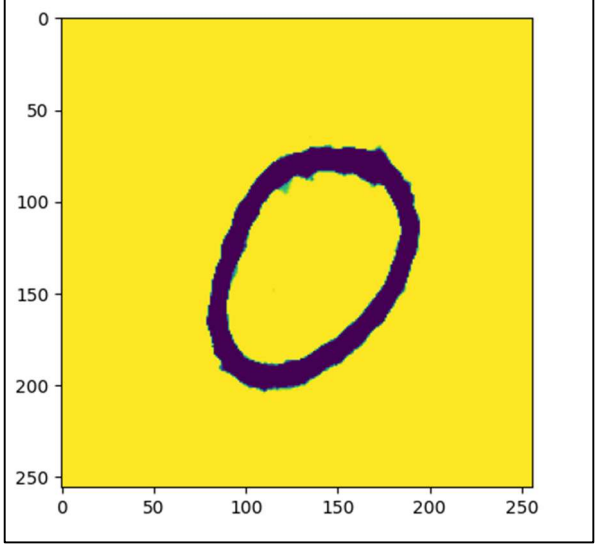

Рисунок 35 – Блок 6 АПД с его выводом

Блок 7 АПД самый большой в данном коде. Здесь происходит отцентровка изображения относительно центра масс пикселей. Блок 7 представлен на рисунке 36.

```
1  #отцентровка изображения по центру масс пикселей
2
3  #изображение загружается в оттенках серого
4  img = cv2.imread('123.png', 0)
5
6  #эта строка находит индексы всех пикселей, значение которых равно
   нулю
7  Y, X = np.nonzero(img == 0)
8  #вычисляет центр масс черных пикселей
9  center_of_mass = (np.mean(X), np.mean(Y))
10
11 #получает размеры изображения
12 height, width = img.shape
13 #вычисление смещения от центра изображения до центра масс
14 shiftX = width//2 - center_of_mass[0]
15 shiftY = height//2 - center_of_mass[1]
16
17 #применяет смещение
18 M = np.float32([[1, 0, shiftX], [0, 1, shiftY]])
19 img = cv2.warpAffine(img, M, (width, height), borderValue=255)
20
21 #выводит изображение
22 cv2_imshow(img)
```

Рисунок 36 – Блок 7 АПД

Здесь строка 7 находит индексы всех пикселей, значение которых равно нулю (то есть черные пиксели в изображении). Направления X и Y заменены для соответствия системе координат изображения, где (0, 0) находится в верхнем левом углу.

Строка 9 вычисляет центр масс черных пикселей, вычисляя среднее значение всех координат X и Y.

Строка 12 получает размеры изображения (высоту и ширину).

Строки 14 и 15 вычисляют смещение от центра изображения до центра масс.

В строке 18 матрица M используется для преобразования изображения. Она представляет собой матрицу преобразования, которая смещает изображение по оси X и Y.

В строке 19 выполняется само преобразование. Метод cv2.warpAffine применяет матрицу преобразования M к изображению, смещая все пиксели. Параметр borderValue=255 гарантирует, что новые пиксели, добавленные в результате смещения, будут белыми.

Строка 22 просто отображает итоговое изображение. Вывод блока представлен на рисунке 37.

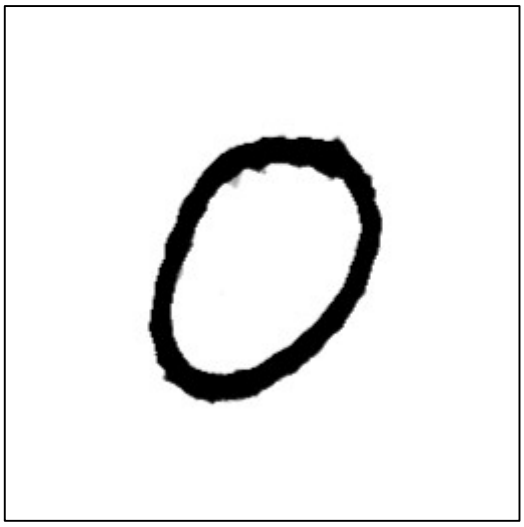

Рисунок 37 – Вывод блока 7 АПД

Последний, восьмой, блок данного блокнота преобразует выходное с предыдущего блока изображение в формат 28x28 пикселей, увеличивает его контраст, инвертирует цвета и сохраняет на диск с заданным названием и расширением. Блок 8 АПД и его вывод на рисунке 38.

Таким образом можно преобразовывать все изображения набора. Итоговый результат преобразований представлен на рисунке 39.

```
img = Image.fromarray(img)
#форматирование изображения и нормализация
img = img.resize((28, 28), Image.ANTIALIAS)
#увеличение контраста
enhancer = ImageEnhance.Contrast(img)
img = enhancer.enhance(2.6)
#инвертирует цвета (белый в чёрный и наоборот)
img = ImageOps.invert(img)
#сохранение изображения в указаной директории
img.save('/content/drive/MyDrive/Colab Notebooks/custom_dataset/Rdy_Img/0.png')
plt.imshow(img)
plt.show()
```

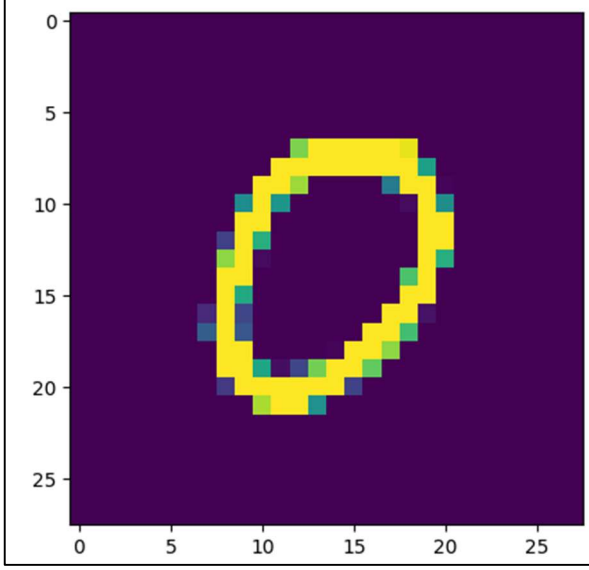

Рисунок 38 – Вывод блока 8 АПД

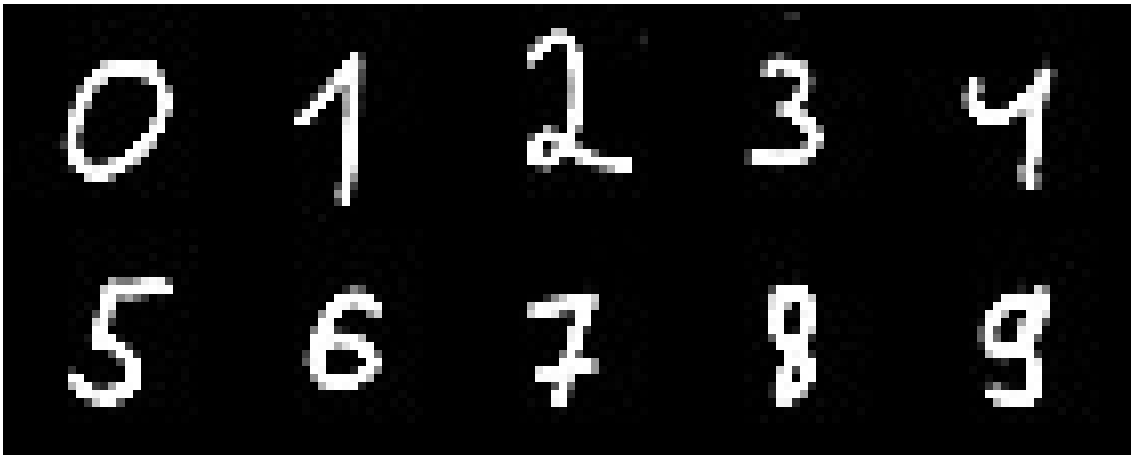

Рисунок 39 – Преобразованные цифры набора

2.1.4 Тестирование и модернизация модели

Когда был получен преобразованный набор цифр, который имеет минимальное количество шумов и соответствует обучающему набору, можно провести тесты на нём для выявления оптимальных параметров, соответствующих этому набору, учитывая, что дальнейшая эксплуатация цифр будет проходить на цифрах, подобных тем, что были на рисунке 20. В таблице 1 представлены тесты структуры ИНС, то есть, в данном случае тестировалось количество скрытых слоёв и нейронов в них, а также количество эпох обучения.

Таблица 1 – Тест структуры ИНС

ИНС										
2 слой	256	256	256	256	256	256	128	512	512	512
3 слой	128	128	128	128	128	128	128	256	256	0
4 слой	0	128	64	64	0	0	0	128	128	0
5 слой	0	0	0	0	0	0	0	0	0	0
выходной слой	10	10	10	10	10	10	10	10	10	10
Эпохи	15	15	15	5	5	15	15	15	5	15
Т.С.Э.	70	70	50	60	50	70	60	50	50	60
Т.Т.Э	97	97	98	98	95	97	97	98	96	97

В таблице 1 Т.С.Э. – это точность в процентах относительно собственных экземпляров, Т.Т.Э. – это точность в процентах относительно тестовых экземпляров из тестового набора. Как видно из таблицы, самыми точными относительно собственных экземпляров структурами для обучения были 784 – 256 – 128 – 10 и 784 – 256 – 128 – 128 – 10 нейронов в соответствующих слоях и 15 эпох обучения. Так как между двумя этими

структурами нет разницы в процентах точности, разумнее будет взять первую из перечисленных структур, так как при той же точности, в ней меньше параметров для обучения и для работы, что делает её оптимальной по времени срабатывания и обучения.

Если обратиться к рисункам 20 и 39, можно заметить остаточные после обработки изображения шумы. А именно, так называемый шум “соль - перец”. Есть вероятность, что подобные и другие шумы могут влиять на выходные результаты модели. Есть способы как обучить ИНС игнорировать такие шумы и лучше распознавать сами паттерны и образы соответствующих цифр. Он заключается в том, чтобы добавить различный шум к обучающему набору изображений. Рассмотрим то, как добавить шум типа “соль - перец” и шум Гаусса к обучающим изображениям, для этого надо модернизировать код синтеза ИНС несколькими блоками.

Первый модернизирующий блок является методом добавления шума “соль - перец” к изображениям. Шум "соль - перец" (salt and pepper noise) — это один из типов аддитивного шума, который может возникать на цифровых изображениях или в сигналах. Он получил свое название благодаря эффекту, когда на изображении появляются случайные белые и черные точки, напоминающие соль и перец. Блок добавляющий этот метод следует за блоком 1 кода синтеза ИНС, поэтому в этой работе его будут называть блок 1.1. По такой аналогии будут названы остальные модернизирующие блоки. Блок 1.1 представлен на рисунке 40.

```
1 #метод добавления шума "соль - перец"
2 def salt_pepper_noise(image, salt_prob, pepper_prob):
3     # Получает размеры изображения
4     row, col = image.shape
5     # Создает копию изображения
6     noisy = image.copy()
7     # Применяет шум "соль"
8     num_salt = np.ceil(salt_prob * image.size)
9     coords = [np.random.randint(0, i - 1, int(num_salt)) for i in image.shape]
10    noisy[coords] = 1
11    # Применяет шум "перец"
12    num_pepper = np.ceil(pepper_prob * image.size)
13    coords = [np.random.randint(0, i - 1, int(num_pepper)) for i in image.shape]
14    noisy[coords] = 0
15    return noisy
```

Рисунок 40 – Блок 1.1

Здесь строка 2 определяет метод, который на вход принимает само изображение, вероятность появления шума “соль” (белых точек) и вероятность появления шума “перец” (черных точек).

Строки 8-10 применяют шум “соль”, где 8 строка считает количество белых точек, которые будут добавлены к изображению, путём умножения заданной вероятности появления на количество пикселей изображения. Затем это значение округляется функцией np.ceil до ближайшего целого значения.

Строка 9 добавляет шум “соль” к случайным координатам изображения.

Строка 10 заменяет пиксели, соответствующие сгенерированным координатам coords, на значение “соль”, которое в данном случае равно 1.

Строки 12-14 применяют шум “перец”, аналогично тому, как применялся шум “соль”, за исключением параметру coords, который равен 0, то есть черным пикселям.

Следующим модернизирующим блоком будет блок метода добавления Гауссовского шума. Он будет располагаться сразу за блоком метода добавления шума “соль - перец” под номером 1.2. Блок 1.2 представлен на рисунке 41.

```
1 #метод добавления Гауссовского шума
2 def add_gaussian_noise(image):
3     #получает размеры изображения
4     row, col = image.shape
5     #коэффициент среднего значения распределения
6     mean = 0.1
7     #коэффициент дисперсии распределения, который определяет степень разброса значений
8     var = 0.2
9     #стандартное отклонение распределения
10    sigma = var ** 0.5
11    #применение шума к изображению
12    gauss = np.random.normal(mean, sigma, (row, col))
13    noisy = np.zeros(image.shape, np.float32)
14    noisy = image + gauss
15    return noisy
```

Рисунок 4 – Блок 1.2

Шум Гаусса характеризуется случайными значениями, распределенными по нормальному (гауссовскому) закону. Этот тип шума добавляет случайные вариации в яркость пикселей изображения. В результате получается некоторое размытие и эффект "пятнистости" на изображении.

Математически шум Гаусса определяется следующим образом: каждое значение пикселя изображения заменяется случайной величиной, которая получается путем добавления случайного значения из нормального распределения с заданным средним значением (обычно 0) и стандартным отклонением (контролирующим интенсивность шума).

На рисунке 41 строки 12-14 применяют шум Гаусса к входному в метод изображению, использую заданные выше параметры и размеры изображения. Строка 12 генерирует случайные значения из нормального (Гауссовского) распределения с использованием функции np.random.normal().

Строка 13 создаёт массив нулей той же формы, что и исходное изображение, с помощью функции np.zeros(). Используется тип данных np.float32 для сохранения точности значений.

В строке 14 применяется шум, складывая каждый пиксель исходного изображения с соответствующим пикселем из шумового массива "gauss". Результат сохраняется в переменной "noisy".

Оптимальные относительно собственного тестового набора изображений параметры, как вероятности появления “соли” и “перца” и коэффициенты среднего значения распределения и дисперсии распределения были найдены опытным путём.

Чтобы применить эти методы на изображения БД MNIST нам понадобиться следующий блок 2.1, который будет выполняться сразу после загрузки изображений из вышеупомянутой БД в блоке 2. Блок 2.1 представлен на рисунке 42.

```
1  #применение метода "соль - перец" для изображений из БД
2  salt_prob = 0.04
3  pepper_prob = 0.02
4
5  x_train_noisy = np.array([salt_pepper_noise(x, salt_prob, pepper_prob) for x in x_train])
6  x_test_noisy = np.array([salt_pepper_noise(x, salt_prob, pepper_prob) for x in x_test])
7
8  #применение метода добавления Гауссовского шума для изображений из БД
9  x_train_noisy = np.array([add_gaussian_noise(x) for x in x_train_noisy])
10 x_test_noisy = np.array([add_gaussian_noise(x) for x in x_test_noisy])
11
12 x_train_noisy = x_train_noisy / 255
13 x_test_noisy = x_test_noisy / 255
```

Рисунок 42 – Блок 2.1

Блок 2.1 применяет методы добавления шума к каждому изображению из массивов x_train и x_test. Строки 12-13 нормализуют значения серости в пределы от 0 до 1, для дальнейшего обучения, аналогично блоку 3, который был описан выше.

Результат работы методов можно визуализировать блоком 4.1, который идёт за блоком 4. Блок 4.1 представлен на рисунке 43. Он работает аналогично блоку 4. Его вывод представлен на рисунке 44.

```
#визуализация изображений из БД с добавленным шумом
plt.figure(figsize=(10,10))
for i in range (25):
  plt.subplot(5,5, i +1)
  plt.xticks([])
  plt.yticks([])
  plt.imshow(x_train_noisy[i], cmap=plt.cm.binary)
  plt.xlabel(class_names[y_train[i]])
```

Рисунок 43 – Блок 4.1

Для наглядности стоит сравнить рисунок 44 с рисунком 11. Как видно на обучающем наборе теперь появился шум. Таким образом ИНС, обученные на таком наборе, будут иметь лучшею обобщающую способность. Такой подход называется “аугментацией данных”.

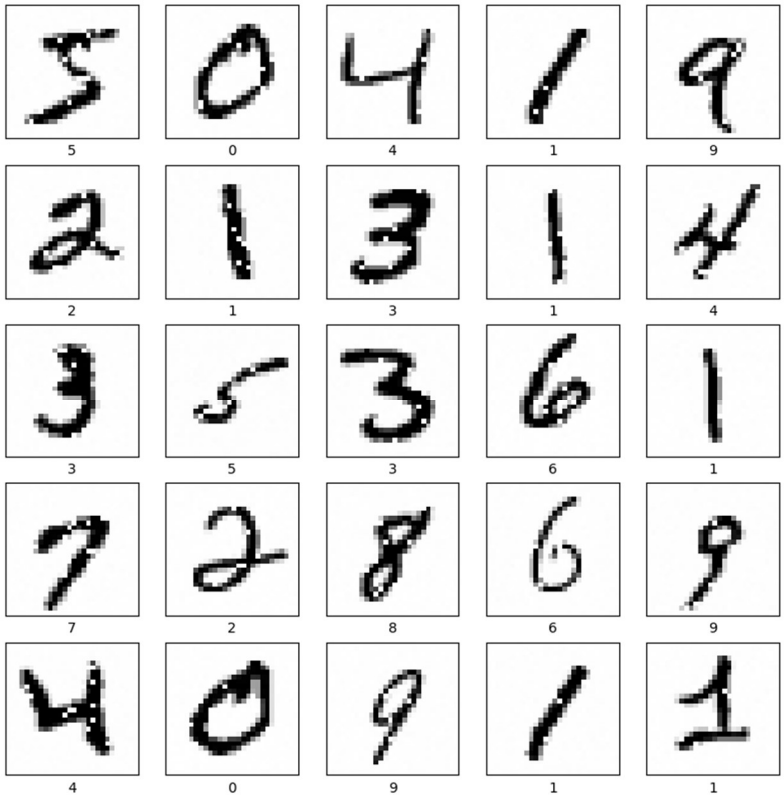

Рисунок 44 – Вывод блока 4.1

Чтобы убедиться в эффективности метода аугментации данных, можно провести тесты модели на собственном наборе данных для ИНС, обученной с этим методом и без. Результаты тестов представлены в таблице 2, где True обозначает, правильно ли модель предсказала класс изображения, η обозначает какую вероятность правильного класса даёт модель в процентах, True sum – это сумма правильных предсказаний, η сред – среднее арифметическое значение показателя η. Метки A и B относятся к модели, обученной без аугментации данных и к модели, обученной с аугментацией соответственно.

Таблица 2 – Результаты тестирования эффективности аугментации данных

Класс	True A	η A, %	True B	η B, %
0	1	64,86	1	92,24
1	1	81,04	1	96,73
2	1	97,9	1	98,04
3	0	0,92	0	9,68
4	1	98,7	1	99,76
5	1	92,16	1	99,89
6	0	10,62	0	3,01
7	1	49,62	1	87,74
8	0	0,74	0	13,04
9	0	13,81	1	88,17
True sum	6		7	
η сред, %		51,04		68,83

Как видно из тестирования, аугментация данных дала прирост точности на 17,79 % и дало на один верный ответ больше. Однако, всё ещё остаются изображения, которые модель не может распознать. В этом заключается основная проблема применения ИНС для распознавания образов рукописных

цифр, так как цифры должны быть очень точно написаны с определённой силой нажатия и точностью. Например, ни одна ИНС за всё время разработки не смогла распознать цифру 6 из собственного набора данных. Чтобы понять в чём проблема, обратимся к рисунку 45.

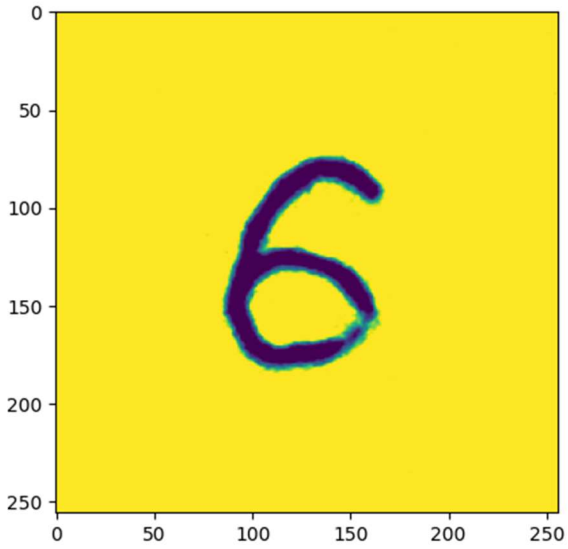

Рисунок 45 – цифра 6 из набора

Из рисунка видно, что овал был написан прерывисто, так как чернил на шарике ручки в тот момент было недостаточно. Из-за этого большая часть протестированных моделей почему-то классифицировали эту цифру как цифру 4 и никакие методы предобработки данных не смоги исправить это. Такие факторы надо учитывать перед применением СРО в своих проектах.

2.1.5 Алгоритм для эксплуатации модели

Финальным алгоритмом проектной части является код, который позволит использовать сохранённую обученную модель. Он будет состоять из 6 блоков. Для ссылки на блоки этого кода будет использована приписка “АИМ” (алгоритм использования модели). Код целиком представлен в приложении В.

Первый блок АИМ подключает библиотеки, большинство из которых уже были рассмотрены выше. Блок 1 АИМ представлен на рисунке 46.

Здесь в строке 2 импорт load_model из модуля tensorflow.keras.models предоставляет функциональность для загрузки предварительно обученных моделей, сохраненных в формате Keras HDF5.

```
1  #позволяет загружать модели
2  from tensorflow.keras.models import load_model
3
4  from google.colab import files
5  import matplotlib.pyplot as plt
6  import numpy as np
7  from PIL import Image
8  import cv2
```

Рисунок 46 – Блок 1 АИМ

Блок 2 АИМ полностью аналогичен блоку 2 АПД на рисунке 30 и просто подключает облачный сервис Google Drive. Для конкретных мест эксплуатации этот блок может отличаться, если, например, на ЭВМ нет доступа к интернету. Хотя в таком случае, этот код также придётся использовать в другой среде программирования, такой как PyCharm.

Блок 3 АИМ загружает модель ИНС из указанной директории. Он представлен на рисунке 47.

```
1  #загружает готовую модель ИНС
2  model = load_model('/content/drive/MyDrive/Colab Notebooks/my_model_Both_v3.h5')
```

Рисунок 47 – Блок 3 АИМ

Блок 4 АИМ загружает обработанное изображение из указанной директории, выводит его на экран и нормализует перед тем, как пустить на вход модели. Блок и его вывод представлены на рисунке 48.

Блок 5 АИМ выводит предсказанную цифру и её вероятность. Блок и его вывод представлены на рисунке 49.

```python
#открывает изображение в указаной директории
img = Image.open ('/content/drive/MyDrive/Colab Notebooks/custom_dataset/Rdy_Img/9.png')

#выводит изображение на экран
plt.imshow(img)
plt.show()

#нормализация и форматирование изображения
img = np.array(img) / 255.0
img = img.reshape(1, 28, 28, 1)
```

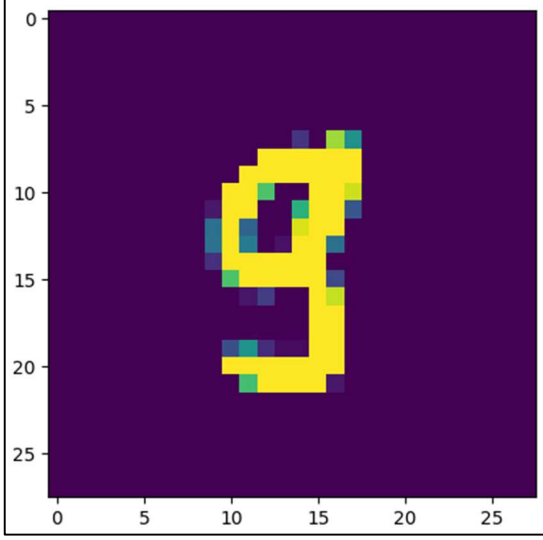

Рисунок 48 – Блок 4 АИМ и его вывод

```python
#выводит предсказанную цифру и указывет вероятность указаной цифры
prediction = model.predict(img)
print("Ответ:", np.argmax(prediction))
print("С веротяностью: {}%".format(round(np.max(prediction)*100, 2)))
```

```
1/1 [==============================] - 0s 280ms/step
Ответ: 9
С веротяностью: 88.12%
```

Рисунок 49 – Блок 5 АИМ и его вывод

Блок 6 АИМ дополнительный, и он выводит вероятности каждого класса для данного изображения. Его можно использовать для тестирования моделей. Блок и вывод на рисунке 50.

```python
#выводит вероятность кажого класса для данного изображения
for value in prediction:
  for index, i in enumerate(value):
    print("Класс:", index)
    print("Вероятность: {}%".format(np.round(i*100, 2)))
    print("--------------------")
```

```
Класс: 0
Вероятность: 0.02%
--------------------
Класс: 1
Вероятность: 6.14%
--------------------
Класс: 2
Вероятность: 0.01%
--------------------
Класс: 3
Вероятность: 0.48%
--------------------
Класс: 4
Вероятность: 3.34%
--------------------
Класс: 5
Вероятность: 1.01%
--------------------
Класс: 6
Вероятность: 0.02%
--------------------
Класс: 7
Вероятность: 0.35%
--------------------
Класс: 8
Вероятность: 0.52%
--------------------
Класс: 9
Вероятность: 88.12%
--------------------
```

Рисунок 50 – Блок 6 АИМ и его вывод

Таким образом можно получать цифровые значения рукописных цифр, сохранять их в переменную и дальше использовать эту информацию. Блок 6 также даёт возможность калибровать модель и алгоритм предобработки данных, основываясь на предсказаниях ИНС.

2.2 Сегментация изображений

Обученная модель ИНС умеет распознавать рукописные цифры в пределах от 0 до 9, чего может быть недостаточно для решения поставленных проблем. Например, если на конвейере больше, чем 10 путей, на которые необходимо отсортировать промаркированный груз. Для решения проблемы

ограниченности определяемых цифр можно использовать сегментацию рукописных цифр. Это подразумевает использование выделения областей для отдельных цифр. Концепт такого метода сегментации представлен на рисунке 51, где можно видеть контейнер с грузом, на котором S1-S4 – сегменты, выделенные для рукописных цифр, U1-U4 – указатели для датчика камеры, соответствующие сегментам S1-S4.

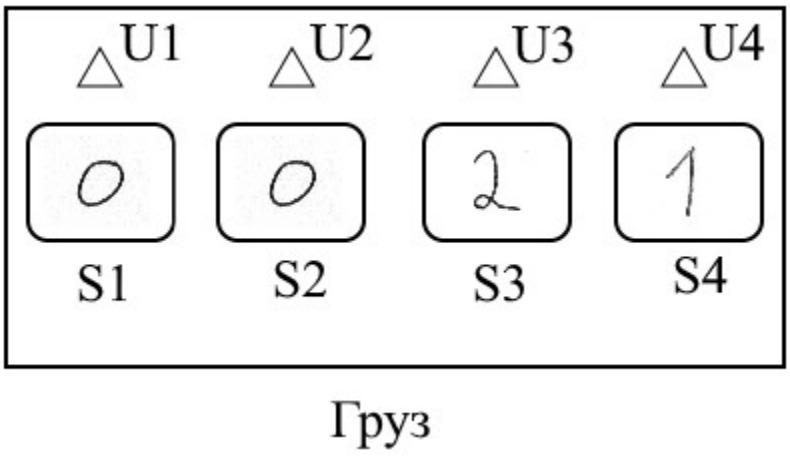

Рисунок 51 – Концепт варианта сегментации

Принцип работы такого концепта заключается в следующем: по конвейеру идёт контейнер с грузом с одной стороной, размеченной как на рисунке 51. Эта сторона должна быть направлена в сторону со считывающей видеокамерой. Указатели для датчика U1-U4 могут быть какими-либо фигурами, имеющими определённый цвет. В таком случае можно использовать считывающую камеру как датчик своего же срабатывания: разделить изображение с камеры на часть для распознавания цифры и часть для ожидания появления цветной фигуры на заданном радиусе или сегменте матрицы камеры. Можно реализовать это так, что форма фигуры не важна и видеокамера будет просто ожидать появления определенного цвета в заданном сегменте с изображения. Также можно использовать цвето-кодировку для

разных сегментов на контейнере, чтобы определённый цвет соответствовал определённому сегменту.

Скажем, что груз двигается слева-направо. Тогда при появлении первого заданного цветом указателя U4 на заданном сегменте матрицы видеокамеры, камера сделает цифровую фотографию. Дальнейший алгоритм предобработки обрежет полученное изображение по заданным пикселям, чтобы осталась только часть с содержимым сегмента S4, соответствующего указателю U4, обработает изображение и даст на вход обученной модели ИНС, которая даст на выходе соответствующий цифре класс, который уже будет записан для сегмента S4. Такая процедура будет проведена для всех сегментов, что, в итоге, даст указанную маркировку груза в цифровом виде.

Однако, такой подход наиболее эффективен в документообороте. Как пример – опросные бланки для получение как-либо цифровой статистики, формы на получение документов, анкеты, квитанции и прочее.

ЗАКЛЮЧЕНИЕ

В дипломной работе был проведён ряд исследований, касающихся искусственного интеллекта, а имена применение и создание искусственных нейронных сетей и их машинное обучение. Это включало в себя изучение областей применения систем распознавания образов на базе машинного обучения, достоинства и недостатки этих систем, изучение структуры искусственных нейронных сетей, их синтез, принцип применения градиентного спуска для обучения нейронных сетей, обратное распространение ошибки, принципы предобработки данных, сбор данных для обучения и прочее.

Также был создан ряд алгоритмов, которые в совокупности представляют из себя систему распознавания образов на базе машинного обучения. Она состоит из написанного кода синтеза моделей искусственных нейронных сетей, кода предобработки данных и кода использования моделей искусственных нейронных сетей. Во время работы были проведены тесты разных структур моделей и определены оптимальные параметры для конкретного набора данных. Для лучшей работы модели была проведена аугментация данных обучающего набора и было проведено исследование, касающееся эффективности этого метода. Был рассмотрен концепт сегментации изображений для расширения использования системы распознавания рукописных цифр.

Из проделанной работы можно сделать следующие выводы: использование ИНС в целом и в распознавании образов на сегодняшний день быстро развивается, что делает актуальным изучение этой технологии. Однако из-за того, что технология только развивается, существуют значительные недостатки в плане применения систем, использующих машинное обучение. Если рассматривать пример распознавания рукописных цифр, основным таким недостатком является нестабильность точности полученной модели ИНС, так

как она требует однотипных обучающему набору данных, чего не всегда возможно добиться. Отсутствие стабильной точности делает затруднительным применение технологии на предприятиях. С другой стороны, СРО крайне стабильно используется в машинах с автопилотом, таких марок как Tesla и BMW [13], [14]. Машины с автопилотом являются отличным примером стабильности применяемой технологии. Так что, вероятно, с большим и более разнообразным набором данных, чем рассмотренная БД MNIST, можно создать более стабильную систему распознавания рукописных цифр. Это заключение приводит к следующей важной проблеме: сбор данных для обучения моделей. Чтобы ИНС работала хорошо, необходим хорошо подобранный огромный набор данных, для сбора которого, уйдёт множество ресурсов, включая время. С развитием сети интернет и увеличением доступной информации это хоть и стало значительно проще, но всё равно такой подход остаётся затратным.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Академик.ру [электронный словарь] – URL: https://www.academic.ru (дата обращения 15.04.2023);

2 Нейрохайв.ио [электронный журнал] – URL: https://www.neurohive.io (дата обращения 15.04.2023);

3 Хабр.ком [электронный журнал] – URL: https://www.habr.com (дата обращения 15.04.2023);

4 Эирпортал.ру [электронный ресурс] – URL: https://www.aiportal.ru (дата обращения 13.03.22);

5 Support vector machines speed pattern recognition – Vision Systems Design. – URL: https://www.vision- systems.com/home/article /16737424/support-vector-machines-speed-pattern-recognition (дата обращения: 20.04.2023);

6 Gangaputra – Handwritten digit database. / Sachin – URL: https://www.cis.jhu.edu/ (дата обращения: 17.08.2013);

7 Zhang Bin; – Fast k – Nearest Neighbor Classification Using Cluster-Based Trees. (англ.) / Sargur N. Srihari / IEEE Transactions on Pattern Analysis and Machine Intelligence (англ.) рус. : journal. — 2004. — Vol. 26, no. 4. — P. 525—528. — doi:10.1109/TPAMI.2004.1265868. — PMID 15382657;

8 Michael Nielsen – Neural Networks and Deep Learning. – 2019. – 6 с.

9 Backpropagation calculus | Chapter 4, Deep learning. [электронный ресурс] – URL: https://www.youtube.com/watch?v=tIeHLnjs5U8&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi&index=5 (дата обращения 19.05.2023)

10 Ying Wen – Handwritten Bangla numeral recognition system and its application to postal automation. – 2007. – 99-107 с.

11 Qisheng Jiang – A Financial Handwritten Digit Recognition Model Based on Artificial Intelligence. – 2023. – 1 с.

12 S.Mahbub-Uz-Zaman – Application of augmented reality: Mobile camera based bangla text detection and translation. / Tanjina Islam – BRAC University. – 2012. – 60 с.

13 Autopilot Tesla [электронный ресурс] – URL: https://www.tesla.com/autopilot (дата обращения 25.04.2023)

14 Autonomous driving [электронный ресурс] – URL: https://www.bmw.com/en/automotive-life/ (дата обращения 25.04.2023)

ПРИЛОЖЕНИЕ А

```
import numpy as np
#используется для работы с массивами

import matplotlib.pyplot as plt
#библиотека для визуализации данных

%matplotlib inline
#команда, позваляющая выводить изображения прямо в блокнот

import tensorflow as tf
#фреймворк для работы и обучения ИНС

from tensorflow import keras
#API для создания и обучения нейронных сетей с использованием фреймворка Keras

from tensorflow.keras.datasets import mnist
#содержит наборы данных для практического использования в обучении моделей

from tensorflow.keras.models import Sequential
#представляет собой последовательную модель нейронной сети

from tensorflow.keras.layers import Dense
#Dense представляет собой полносвязный слой нейронной сети

#######################################
### библиотеки для работы с данными ###
#######################################

import cv2
#библиотека алгоритмов компьютерного зрения и обработки изображений.

from PIL import Image
#основной класс для работы с изображениями

from PIL import ImageOps
#модуль, позволяющий конвертацию изображения в градации серого

from PIL import ImageEnhance
# модуль, содержащий классы для улучшения изображений

from google.colab import files
#модуль, предоставляющий методы для загрузки и скачивания файлов

from google.colab import drive
#модуль, предоставляющий методы для подключения к Google Drive

import glob
#позволяет считывать названия файлов по шаблонам

import os
#позволяет управлять файлами и директориями

from tensorflow.keras.models import load_model
#позволяет загружать модели


#метод добавления шума "соль - перец"
def salt_pepper_noise(image, salt_prob, pepper_prob):
    # Получает размеры изображения
    row, col = image.shape
    # Создает копию изображения
    noisy = image.copy()
    # Применяет шум "соль"
    num_salt = np.ceil(salt_prob * image.size)
```

```
            coords = [np.random.randint(0, i - 1, int(num_salt)) for i in image.shape]
            noisy[coords] = 1
            # Применяет шум "перец"
            num_pepper = np.ceil(pepper_prob * image.size)
            coords = [np.random.randint(0, i - 1, int(num_pepper)) for i in image.shape]
            noisy[coords] = 0
            return noisy

        #метод добавления Гауссовского шума
        def add_gaussian_noise(image):
            #получает размеры изображения
            row, col = image.shape
            #коэффициент среднего значения распределения
            mean = 0.1
            #коэффициент дисперсии распределения
            var = 0.2
            #стандартное отклонение распределения
            sigma = var ** 0.5
            #применение шума к изображению
            gauss = np.random.normal(mean, sigma, (row, col))
            noisy = np.zeros(image.shape, np.float32)
            noisy = image + gauss
            return noisy
        #загрузка изображений из БД MNIST
        (x_train, y_train), (x_test, y_test) = mnist.load_data()
        #добавление названий классов
        class_names = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']

        #применение метода "соль - перец" для изображений из БД
        salt_prob = 0.04
        pepper_prob = 0.02

        x_train_noisy = np.array([salt_pepper_noise(x, salt_prob, pepper_prob) for x in
x_train])
        x_test_noisy = np.array([salt_pepper_noise(x, salt_prob, pepper_prob) for x in
x_test])

        #применение метода добавления Гауссовского шума для изображений из БД
        x_train_noisy = np.array([add_gaussian_noise(x) for x in x_train_noisy])
        x_test_noisy = np.array([add_gaussian_noise(x) for x in x_test_noisy])

        x_train_noisy = x_train_noisy / 255
        x_test_noisy = x_test_noisy / 255

        #изменение значений серого в пределы от 0 до 1
        x_train = x_train / 255
        x_test = x_test / 255

        #визуализация изображений из БД
        plt.figure(figsize=(10,10))
        for i in range (25):
          plt.subplot(5,5, i +1)
          plt.xticks([])
          plt.yticks([])
          plt.imshow(x_train[i], cmap=plt.cm.binary)
          plt.xlabel(class_names[y_train[i]])

        #визуализация изображений из БД с добавленным шумом
        plt.figure(figsize=(10,10))
        for i in range (25):
          plt.subplot(5,5, i +1)
          plt.xticks([])
          plt.yticks([])
          plt.imshow(x_train_noisy[i], cmap=plt.cm.binary)
          plt.xlabel(class_names[y_train[i]])
```

```
        #вызов метода создания ИНС с заданием кол-ва нейронов и слоев
        model = keras.Sequential ([
                                   keras.layers.Flatten(input_shape=(28,28)),
                                   keras.layers.Dense (256, activation="relu"),
                                   keras.layers.Dense (128, activation="relu"),
                                   keras.layers.Dense (10, activation="softmax")
                                  ])

        #вызов метода обучения ИНС
        model.compile(optimizer=tf.keras.optimizers.SGD(),
                      loss='sparse_categorical_crossentropy',
                      metrics=['accuracy'])

        #визуализация параметров ИНС
        model.summary()

        #вызов метода обучения ИНС по эпохам
        model.fit(x_train_noisy, y_train, epochs=15)

        #выводит значение точности модели в отношении к тестовому набору
        test_loss, test_acc = model.evaluate(x_test_noisy, y_test)
        print('Точность теста: ', test_acc)


        #подключает облачный диск Google Drive
        #drive.mount('/content/gdrive')

        #Сохраняет модель на облачном диске
        #model.save('/content/gdrive/MyDrive/Colab Notebooks/my_model_clean.h5')

        #загружает готовую модель ИНС
        model = load_model('/content/drive/MyDrive/Colab Notebooks/my_model_70per.h5')

        #пример создания собственного набора тестовых данных

        #в этой матрице будут храниться массивы изображений
        image_list = []
        #в этом векторе будут храниться метки для каждого изображения
        label_list = []

        #цикл обработки и добавления в набор каждого изображения типа .png
        for filename in glob.glob('/content/drive/MyDrive/Colab
Notebooks/custom_dataset/Rdy_Img/*.png'):

            #добавление изображения в переменную im
            im = Image.open(filename)
            #конвертация изображения в оттенки серого
            im = im.convert('L')
            #увеличение контраста для снижения шумов
            enhancer = ImageEnhance.Contrast(im)
            im = enhancer.enhance(2)
            #сохранение изображения и конвертация в массив
            im.save ('/content/123.png')
            im = cv2.imread('/content/123.png', 0)
            #добавление медианного размытия для снижения шумов
            im = cv2.medianBlur(im,1)
            #сохранение изображения из массива
            im = Image.fromarray(im)
            #повторное увелечение контраста
            enhancer = ImageEnhance.Contrast(im)
            im = enhancer.enhance(8.2)
            #изменение изображений в нужное разрешение с применением сглаживания
            im = im.resize((28, 28), Image.ANTIALIAS)
            enhancer = ImageEnhance.Contrast(im)
            im = enhancer.enhance(2.6)
```

```
    #инвертирует цвета (белый в чёрный и наоборот)
    im = ImageOps.invert(im)

    #нормализация
    im = np.array(im) / 255.0

    # получает имя файла
    base = os.path.basename(filename)
    # разделяет имя файла на имя и расширение и преобразуем имя в целое число
    label = int(os.path.splitext(base)[0])
    label_list.append(label)
    image_list.append(im)


X = np.array(image_list)
Y = np.array(label_list)

#визуализация изображений из собственного набора
plt.figure(figsize=(10,10))
for i in range (10):
  plt.subplot(5,5, i +1)
  plt.xticks([])
  plt.yticks([])
  plt.imshow(X[i], cmap=plt.cm.binary)
  plt.xlabel(class_names[Y[i]])

test_loss, test_acc = model.evaluate(X, Y)
print('Точность теста: ', test_acc)
# Предсказание вероятностей классов для тестового набора
predictions = model.predict(X)

# Печать вероятностей
for i, prediction in enumerate(predictions):
    print(f'Изображение {i+1}, метка класса: {Y[i]}')
    print("Предсказание класса:", np.argmax(prediction))
    print("С веротностью: {}%".format(round(np.max(prediction)*100, 2)))
    print('----------------------------------')
```

ПРИЛОЖЕНИЕ Б

```
import cv2
import numpy as np
from google.colab import drive
from google.colab import files
from PIL import Image
from PIL import ImageOps
from PIL import ImageEnhance
import matplotlib.pyplot as plt
%matplotlib inline

#позволяет просматривать изображения cv2 в блокноте
from google.colab.patches import cv2_imshow

#подключает облачный диск Google Drive
#drive.mount('/content/gdrive')

#загрузка изображения в переменную img
img = Image.open('/content/drive/MyDrive/Colab Notebooks/custom_dataset/6.png')
#конвертация изображения в оттенки серого
img = img.convert('L')
#увеличение контраста
enhancer = ImageEnhance.Contrast(img)
img = enhancer.enhance(2)
#сохранение изображения для последующего открытия в OpenCV
img.save('123.png')
#вывод изображения в блокнот
plt.imshow(img)
plt.show()

#открытие изображения в OpenCV и добавление медианного размытия для снижения шумов
img = cv2.imread('/content/123.png', 0)
img = cv2.medianBlur(img,5)
cv2_imshow(img)

#обратное сохранение изображения для обработки с помощью PLT
img = Image.fromarray(img)
plt.imshow(img)
plt.show()

#увеличение контраста
enhancer = ImageEnhance.Contrast(img)
img = enhancer.enhance(8.2)
img.save('123.png')
plt.imshow(img)
plt.show()
```

```
#отцентровка изображения по центру масс пикселей

#изображение загружается в оттенках серого
img = cv2.imread('123.png', 0)

#эта строка находит индексы всех пикселей, значение которых равно нулю
Y, X = np.nonzero(img == 0)
#вычисляет центр масс черных пикселей
center_of_mass = (np.mean(X), np.mean(Y))

#получает размеры изображения
height, width = img.shape
#вычисление смещения от центра изображения до центра масс
shiftX = width//2 - center_of_mass[0]
shiftY = height//2 - center_of_mass[1]

#применяет смещение
M = np.float32([[1, 0, shiftX], [0, 1, shiftY]])
img = cv2.warpAffine(img, M, (width, height), borderValue=255)

#выводит изображение
cv2_imshow(img)

img = Image.fromarray(img)
#форматирование изображения и нормализация
img = img.resize((28, 28), Image.ANTIALIAS)
#увеличение контраста
enhancer = ImageEnhance.Contrast(img)
img = enhancer.enhance(2.6)
#инвертирует цвета (белый в чёрный и наоборот)
img = ImageOps.invert(img)
#сохранение изображения в указаной директории
img.save('/content/drive/MyDrive/Colab Notebooks/custom_dataset/Rdy_Img/6.png')
plt.imshow(img)
plt.show()
```

ПРИЛОЖЕНИЕ В

```
#позволяет загружать модели
from tensorflow.keras.models import load_model

from google.colab import files
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
import cv2

#подключает облачный диск Google Drive
#drive.mount('/content/gdrive')

#загружает готовую модель ИНС
model = load_model('/content/drive/MyDrive/Colab Notebooks/my_model_Both_v3.h5')

#открывает изображение в указаной директории
img = Image.open ('/content/drive/MyDrive/Colab
Notebooks/custom_dataset/Rdy_Img/9.png')

#выводит изображение на экран
plt.imshow(img)
plt.show()

#нормализация и форматирование изображения
img = np.array(img) / 255.0
img = img.reshape(1, 28, 28, 1)

#выводит предсказанную цифру и указывет вероятность указаной цифры
prediction = model.predict(img)
print("Ответ:", np.argmax(prediction))
print("С веротяностью: {}%".format(round(np.max(prediction)*100, 2)))

#выводит вероятность кажого класса для данного изображения
for value in prediction:
  for index, i in enumerate(value):
    print("Класс:", index)
    print("Вероятность: {}%".format(np.round(i*100, 2)))
    print("--------------------")
```

ПРИЛОЖЕНИЕ Г