МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет вычислительной техники

Кафедра «Вычислительная техника»

Направление подготовки: 09.04.01 «Информатика и вычислительная техника»

Магистерская программа: «Прикладной искусственный интеллект»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему

ИСПОЛЬЗОВАНИЕ НЕЙРОННЫХ СЕТЕЙ ДЛЯ РАСПОЗНАВАНИЯ ОБЪЕКТОВ НА ИЗОБРАЖЕНИЯХ

Студент 13.06.24 (подпись, дата) Емелин Сергей Владимирович (ФИО полностью)

Руководитель 13.06.24 (подпись, дата) Карамышева Н. С. (фамилия, инициалы)

Нормконтролёр 13.06.24 (подпись, дата) Бычков А.С. (фамилия, инициалы)

Рецензент (подпись, дата) 17.06.2024 (фамилия, инициалы) (должность, место работы)

Работа допущена к защите (протокол зас. кафедры от 27.05.2024 г. № 11)

Заведующий кафедрой (подпись) Митрохин М. А. (фамилия, инициалы)

Работа защищена с отметкой хорошо (протокол заседания ГЭК от 18.06.24 № 25)

Секретарь ГЭК (подпись) Гурин Е. И. (фамилия, инициалы)

Пенза 2024

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой

«Утверждаю»
Заведующий кафедрой
_____ М.А. Митрохин
«13» 09 2022 г.

ЗАДАНИЕ

НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ

1. Студент Емелин Сергей Владимирович гр. 22ВВИм1 факультета вычислительной техники, направления 09.04.01.

2. Тема работы «Использование нейронных сетей для распознавания объектов на изображениях».

Тема утверждена приказом ПГУ № 765/о от "13"сентября 2022 г.

3. Руководитель магистерской диссертации к.т.н., доцент кафедры ВТ ПГУ Карамышева Н. С.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Назначение разработки: упрощение сравнения точности архитектурных нейросетевых решений в задаче распознования обьекта на изображении.

Разработать и внедрить программное обеспечение, позволяющее сравнить точность нейросетевых решений в задаче распознования обьектов на изображении.

5. Перечень подлежащих разработке вопросов:

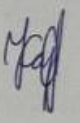
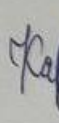
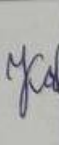
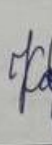
1) Создание графического интерфейса для приложения реализующее распознования цифр на изображении.

2) Реализация нейросетевой модели.

3) Обучение нейросетевой модели.

4) Апробация реализованного функционала в форме тестирования.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучить теоретические основы существующих реализаций распознования обьектов на изображении с помощью нейросетей	10%	27.09.2022-15.11.2022	
2.	Анализ существующих методов машинного обучения для распознования объектов на изображении	10%	16.11.2022-22.03.2023	
3.	Проектирование архитектуры приложения	10%	23.03.2023-14.06.2023	
4.	Подготовка обучающих данных для нейросетевой модели	10%	15.06.2023-05.01.2024	
5.	Разработка программного обеспечения	40%	06.01.2024-27.04.2024	
6.	Отладка системы	10%	28.04.2024-10.05.2024	
7.	Оформление выпускной работы	10%	11.05.2024-28.05.2024	

Дата выдачи задания "13" сентября 2022 г.

Руководитель магистерской диссертации 13.09.2022 Карамышева Н.С.
(подпись, дата)

Задание к исполнению принял студент 13.09.2022 Емелин С.В.
(подпись, дата)

Работу к защите допустить
Декан факультета 17.06.2024 Фионова Л.Р.
(подпись, дата)

РЕФЕРАТ

Магистерская работа содержит 105 страниц, 26 рисунков, 10 источников, 2 приложения, 6 формул, 2 таблицы

МАШИННОЕ ОБУЧЕНИЕ, НЕЙРОСЕТИ, СВЕРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ, ТРАНСФОРМЕРЫ, КОМПЬЮТЕРНОЕ ЗРЕНИЕ, РАСПОЗНОВАНИЕ ОБЪЕКТОВ

Цель работы – является создание приложения с пользовательским интерфейсом, которое позволит пользователям рисовать цифры и выбор какую из двух архитектур использовать для их распознавания.

Объектом разработки является интерактивное приложение для распознавания рукописных цифр. Данная система позволяет пользователю рисовать цифры и выбирать между сверточными нейронными сетями и трансформерами для их распознавания, предоставляя немедленную обратную связь о предсказанных цифрах и уровне уверенности моделей.

В результате проведенной работы с нейросетевой архитектуры сверточных нейронных сетей и трансформера был реализован приложение для распознавания рукописных цифр.

Результаты данной работы могут быть применены в области классификации изображений, автоматической идентификации и категоризации визуальных данных.

СОДЕРЖАНИЕ

Перечень Сокращений и обозначений 4
Введение 5
1 Актуальность и цели исследования 6
1.1 Актуальность 6
1.2 Постановка задачи 7
2 Используемые методы распознавания цифр на изображении 9
2.1 Сверточные нейронные сети 9
2.1.1 Архитектура сверточных нейронных сетей 11
2.1.2 Обучение сверточных нейронных сетей 13
2.1.3 Преимущества в распознавании рукописных цифр: 16
2.1.4 Ограничения сверточных нейронных сетей в распознавании рукописных цифр: 17
2.2 Vision Transformer (ViT) 18
2.2.1 Архитектура ViT 19
2.2.2 Преимущества ViT 20
2.2.3 Обучение ViT 20
3 Разработка программного обеспечения 21
3.1 Выбор языка программирования 21
3.2 Используемые библиотеки 22
3.3 Описание работы итоговой нейронной сети и замеры точности 24
3.4 Описание кода 42
4 Результаты тестирования 48
4.1 Успешные распознования 48
4.2 Ошибочные распознования 59
Заключение 62
Список использованных источников 63

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей магистерской диссертации применяются следующие сокращения и обозначения

МЛФ - многопеременная логистическая функция

ПО – программное обеспечение

CNN (Convolutional Neural Network) – свёрточная нейронная сеть

C++ – язык программирования

PyQt5 – набор Python библиотек для создания графического интерфейса

Python – язык программирования

ВВЕДЕНИЕ

В процессе развития человечества все чаще поднимается вопрос об автоматизации каких либо процессов и благодаря технологиям, автоматизация все больше и больще проникает в нашу повседневную жизнь. С появлением искуственного интеллекта, специализированного оборудования становится проще решать типовые задачи.

Одной из таких задач является оптическое распознавание символов (OCR), в том числе распознавание рукописного текста. Одной из важных проблем в рукописном тексте, это отсутствие стандарта написания, в отличии от печатного, каждый человек имеет свой подчерк. Цифры имеют наклон, лишние завитки или черточнки, что цифру не меняет, но делает ее уникальной в написании.

В данной работе разрабатывается программное обеспечение (ПО) которое позволит пользователю рисовать цифры в графическом интерфейсе и изучается распознование математических цифр с помощью искуственного интеллекта.

Существует два типа распознавания рукописных символов: online и offline распознавание. Online распознавание например ипользуется в сенсорных экранах. ПО отслеживает движения пальца пользователя, когда он рисует символы, далее эта информация участвует в алгоритме определения символа написанного пользователем.

В offline распозновании доступна только конечная информация, к примеру различные сканы рукописных документов. Этот тип сложнее за счет того, что неизвестно как автор писал символы, подчерк, стилевые особенности, низкое качество рассматриваемого изображения,

Решение задачи offline распознавания математических цифр и будет рассмотрено в этой работе.

1 Актуальность и цели исследования

1.1 Актуальность

На сегодняшний день существует ряд систем, которые значительно упрощают жизнь инженерам и преподавателям естественных наук. Наиболее подходящими для данного иследования являются такие программы как Wolfram Mathematica, Matlab. Но они имеют крупный недостаток в необходимости знания и умения оперировать специальным синтаксисом, необходимим для решения задач. Было бы удобнее использовать приложение, способное считывать изображение с написанной от руки цифрой. Такие решения уже существуют и активно применяются.

Рассмотрим первый такой сервис: Google Cloud Vision API. Это API анализирует содержимое изображений с помощью алгоритмов машинного обучения, обучающиеся на огромных объемах данных. Это является ключевым аспектом, поскольку гарантирует точное распознавание текста и объектов на изображениях. Программное обеспечение умеет распозновать текст, лица, объекты что позволяет использовать его для широкого спектра задач обработки изображений. Это облачный сервис, поэтому проблема поддержки собтсвенной инфраструктуры отпадает, а для интеграции очень прост, так как поддерживает множество языков программирования. Однако у Google Cloud Vision API есть и недостатки. Во-первых, это платная услуга, и использование API может быть дорогостоящим, особенно при обработке большого объема изображений или при использовании продвинутых функций. Возможности настройки алгоритмов ограничены по сравнению с созданием собственной модели машинного обучения, что может не соответствовать специфическим потребностям. Кроме того, обработка данных в облаке может вызвать определенные требования к безопасности и конфиденциальности данных, особенно если изображения содержат конфиденциальную информацию.

Второй сервис для рассмотрения будет Microsoft Azure Computer Vision. который имеет множество функций для распознавания и анализа данных и так

же умеет распозновать текст, объекты, лица и цвета. ПО позволяет обрабатывать в реальном времени большие объемы данных, а также предлагает простой api, который упрощает разработку и интеграцию. В недостатках Microsoft Azure Computer Vision можно выделить то, что это платный сервис, и чем больше будет данных для обработки, тем выше цена и при использовании продвинутых функций и моделей, так же повышается цена. Возможности настройки алгоритмов и моделей могут быть ограничены по сравнению с созданием собственных моделей машинного обучения. Платформа также требует постоянного доступа к интернету и зависит от доступности сервиса Azure, поскольку работает в облачной среде.

Третим рассмотриваемым сервисом получился Amazon Textract. Это высокоэффективный сервис извлечения текста и информации с изображений, который распознает и текст и рукописные цифры, что делает его полезным инструментом для автоматизированного извлечения данных из различных типов документов и изображений. Amazon Textract позволяет обрабатывать большие объемы изображений и документов без потери производительности. Простота и интуитивно понятный интерфейс для работы с API упрощают процесс разработки и интеграции.

Недостатком Amazon Textract так же являетя, то что он платный. Это может неподойти для ряда компаний и сфер. Как и другие облачные сервисы, Amazon Textract требует постоянного доступа к интернету и зависит от доступности инфраструктуры AWS.

1.2 Постановка задачи

Основной целью является создание приложения с пользовательским интерфейсом, которое позволит пользователям рисовать цифры и выбор какой метод распознавания использовать. Для этого нужно использовать

предварительно обученную нейронную сеть, способную распознавать рукописные цифры. В конечном итоге цель предоставить пользователям результаты распознавания в удобном формате.

Для достижения цели разобьем проект на несколько задач. Во-первых, нужно реализовать пользовательский интерфейс, обеспечивающий удобный доступ к функциям приложения. Затем следует загрузить предварительно обученную модель распознавания рукописных цифр для использования в приложении. Далее необходимо разработать функциональность, позволяющую рисовать цифры и передавать пользовательские изображения в модель для распознавания. Также нужно создать механизм для отображения результатов распознавания нарисованных цифр, чтобы пользователи могли видеть результаты своей работы. Наконец, оптимизируем интерфейс и функциональность приложения, чтобы обеспечить более удобное и интуитивно понятное взаимодействие с пользователем.

Ожидается, что разработанное приложение будет успешно распознавать рукописные цифры с высокой точностью. Пользователи смогут получать быстрые результаты их распознавания. Интерфейс приложения будет удобным и интуитивно понятным для использования.

Важно учитывать следующее ограничение - производительность приложения может ограничиваться вычислительными ресурсами устройства пользователя.

2 Используемые методы распознавания цифр на изображении

2.1 Сверточные нейронные сети

Для решения поставленной задачи можно применить несколько методов и алгоритмов, в том числе использование сверточных нейронных сетей (CNN) и трансформер. Эти сети представляют собой специализированный тип нейронных сетей, которые эффективно обрабатывают данные с пространственной структурой, такие как изображения.

Сверточные нейронные сети состоят из нескольких слоев, включая сверточные слои, слои подвыборки и полносвязные слои. Сверточные слои выполняют операцию свертки, выделяющую важные признаки изображения. Слои подвыборки уменьшают размерность данных, сохраняя ключевые признаки, в то время как полносвязные слои классифицируют и определяют, к какому классу принадлежит рукописная цифра.

Обучение сверточных нейронных сетей для распознавания рукописных цифр проходит через два основных этапа: обучение и тестирование. На этапе обучения сеть обрабатывает обширный набор размеченных данных, чтобы научиться распознавать различные цифры. В процессе тестирования сеть оценивается на новых, не размеченных данных, чтобы определить ее точность и эффективность.

Процесс распознавания рукописных цифр с использованием сверточных нейронных сетей начинается с подачи изображения рукописной цифры на вход сети, последующим прямым распространением сигнала через ее слои, получением выходных данных сети в виде вероятностей принадлежности к каждому классу цифр и выбором класса с наибольшей вероятностью.

Сверточные нейронные сети обладают рядом преимуществ в распознавании рукописных цифр, таких как высокая точность и способность обрабатывать изображения с различными размерами и ориентациями. Однако они также имеют некоторые ограничения, такие как необходимость большого

объема размеченных данных для обучения и высокие вычислительные требования.

Примеры успешного применения сверточных нейронных сетей в распознавании рукописных цифр включают систему распознавания рукописных цифр MNIST и систему распознавания рукописных цифр на мобильных устройствах.

Сверточные нейронные сети (CNN) представляют собой разновидность нейронных сетей, специально созданных для анализа данных, представленных в виде двухмерных или многомерных сеток, таких как изображения. Основная концепция CNN заключается в использовании операции свертки для извлечения локальных характеристик из входных данных. Эта операция представляет собой поэлементное умножение двух матриц и последующее суммирование результатов. В контексте CNN свертка применяется к входным данным с использованием фильтров (ядер), которые перемещаются по входу и вычисляют локальные характеристики.

Архитектура CNN включает несколько слоев, таких как сверточные слои, слои пулинга и полносвязные слои. Сверточные слои используются для извлечения признаков, слои пулинга уменьшают размерность данных, а полносвязные слои выполняют классификацию или регрессию на основе извлеченных признаков.

CNN имеют несколько преимуществ перед традиционными нейронными сетями при работе с изображениями. Они автоматически извлекают иерархические характеристики из данных, что позволяет им эффективно обрабатывать изображения разных размеров и ориентаций. Кроме того, CNN учитывают локальные связи между пикселями, что делает их более эффективными в обработке изображений.

Кроме того, сверточные нейронные сети могут проводить параметрическое обучение. Это позволяет им автоматически настраивать параметры своих фильтров в процессе обучения, опираясь на минимизацию функции потерь. Эта

способность обеспечивает им гибкость в адаптации к различным данным и задачам, избегая необходимости ручной настройки параметров. Для повышения эффективности и скорости обучения многие сверточные нейронные сети используют дополнительные методы, такие как пакетная нормализация и регуляризация, которые стабилизируют процесс обучения и делают модели более устойчивыми к переобучению. Эти преимущества позволяют широко применять сверточные нейронные сети в различных областях, включая компьютерное зрение, распознавание образов, медицинскую диагностику, автоматическое распознавание речи и другие. Их успешное использование в таких задачах подтверждает их высокую эффективность и универсальность в анализе и обработке различных типов данных, представленных в форме двумерных или многомерных сеток.

2.1.1 Архитектура сверточных нейронных сетей

Архитектура сверточных нейронных сетей (CNN) играет ключевую роль в анализе и обработке изображений. Она определяет, как данные передаются через сеть и как они обрабатываются для решения конкретной задачи, такой как распознавание рукописных цифр или классификация изображений.

Сверточные слои (Convolutional layers): Основной строительный блок CNN. Каждый сверточный слой состоит из набора фильтров или ядер, которые применяются к входным данным. Фильтры скользят по входным данным, выполняя операцию свертки, которая позволяет извлекать локальные признаки изображения. Результатом свертки является карта признаков, которая представляет собой активации фильтров на различных местоположениях входных данных.

Пулинговые слои (Pooling layers): Используются для уменьшения размерности карты признаков. Это помогает уменьшить вычислительную сложность и количество параметров модели, а также сделать модель более устойчивой к переобучению. Один из наиболее распространенных типов

пулинга - операция максимального пулинга, которая выбирает максимальное значение из каждой области карты признаков.

Полносвязные слои (Fully connected layers). После извлечения признаков сверточными и пулинговыми слоями данные передаются полносвязным слоям. В этих слоях каждый нейрон соединен со всеми нейронами предыдущего слоя. Полносвязные слои обычно используются для классификации или принятия решений на основе извлеченных признаков.

Функции активации (Activation functions). Применяются к выходам каждого нейрона в сети и определяют, должен ли нейрон активироваться или нет. Они придают сети нелинейность, что позволяет ей моделировать сложные взаимосвязи в данных. Некоторые популярные функции активации включают ReLU (Rectified Linear Unit), сигмоиду и гиперболический тангенс.

Архитектура CNN может содержать различные комбинации сверточных, пулинговых и полносвязных слоев в зависимости от конкретной задачи и требований. Обычно она начинается с нескольких сверточных слоев для извлечения признаков изображения, за которыми следуют пулинговые слои для уменьшения размерности, а затем один или несколько полносвязных слоев для классификации или решения других задач.

Еще одним важным компонентом архитектуры CNN является слой нормализации, который применяется для стабилизации и ускорения процесса обучения путем нормализации активаций внутри сети. Этот слой может включать различные методы, например, пакетную нормализацию, которая нормализует активации в каждом пакете данных во время обучения.

Для преодоления проблемы переобучения, возникающей при обучении на небольших наборах данных, часто используются методы регуляризации, такие как Dropout, который случайным образом исключает нейроны во время обучения, предотвращая слишком сильное подгонку сети под обучающие данные.

Существует множество различных архитектур CNN, каждая из которых имеет свои особенности и применение. Например, архитектура ResNet известна своей способностью обучаться на глубоких сетях без проблем с затуханием градиента, в то время как сеть U-Net часто используется для сегментации изображений, особенно в медицинских приложениях.

В общем, архитектура CNN представляет собой сложную комбинацию различных слоев и методов, которые совместно обеспечивают эффективный анализ и обработку изображений, делая их важным инструментом в области компьютерного зрения и машинного обучения.

2.1.2 Обучение сверточных нейронных сетей

Обучение сверточных нейронных сетей (СНС) является процессом, в ходе которого сеть обучается распознавать и классифицировать изображения. Этот процесс включает в себя два основных этапа: прямое распространение и обратное распространение ошибки.

Прямое распространение: во время прямого распространения, входные данные (изображение) подается на вход сверточной нейронной сети. Изображение проходит через сверточные слои, пулинг слои и полносвязные слои, где каждый слой выполняет определенные операции над данными. На выходе получается предсказание сети, которое может быть в виде вероятностей для каждого класса или конкретного значения.

Обратное распространение ошибки: после прямого распространения, сравнивается предсказание сети с правильным ответом (меткой класса) для данного изображения. Разница между предсказанием и правильным ответом выражается в виде функции потерь, которая измеряет ошибку сети. Затем, с помощью алгоритма обратного распространения ошибки, эта ошибка распространяется назад через сеть, позволяя ей корректировать веса и параметры своих нейронов.

В процессе обратного распространения, градиенты ошибки вычисляются для каждого параметра сети, и эти градиенты используются для обновления

весов и параметров с помощью метода оптимизации, такого как стохастический градиентный спуск (SGD) или его вариации.

Обучение сверточных нейронных сетей требует большого объема размеченных данных для эффективного обучения. Чем больше данных доступно для обучения, тем лучше сеть сможет обобщать и делать точные предсказания на новых, ранее не виденных данных.

После завершения процесса обучения, сверточная нейронная сеть может быть использована для классификации новых изображений, предсказывая их класс или конкретное значение.

Обучение сверточных нейронных сетей - это важный этап, определяющий эффективность и точность модели в дальнейшем прогнозировании. Важно отметить, что процесс обучения может занимать значительное время и требует наличия мощных вычислительных ресурсов, особенно при использовании больших наборов данных и глубоких архитектур сетей.

Помимо этого, процесс обучения может быть дополнен различными техниками и стратегиями для улучшения результатов. Например, использование аугментации данных может увеличить разнообразие тренировочного набора путем применения случайных преобразований к изображениям, таких как повороты, сдвиги, изменения масштаба и т. д. Это позволяет модели обучаться на более разнообразных данных и делает ее более устойчивой к различным условиям.

Также важно проводить мониторинг процесса обучения и анализировать метрики эффективности модели на валидационном наборе данных. Это помогает выявить возможные проблемы, такие как переобучение или недообучение, и корректировать параметры модели или алгоритмы оптимизации в соответствии с наблюдаемыми результатами.

Наконец, обучение сверточных нейронных сетей является итеративным процессом, и часто требуется несколько циклов обучения и настройки параметров, прежде чем достигнутся желаемые результаты. Это означает, что

исследователи и разработчики должны быть готовы к тщательному анализу и постоянному совершенствованию моделей для достижения оптимальных результатов в задачах классификации и распознавания изображений.

По окончании обучения сверточной нейронной сети она проходит тестирование на наборе данных, не использованных в процессе обучения. Тестовые изображения подаются на вход, а сеть делает предсказания для каждого из них. После этого предсказания сравниваются с правильными ответами, и вычисляются метрики, такие как точность, для оценки производительности сети.

Таким образом, процесс распознавания рукописных цифр с помощью сверточных нейронных сетей включает подготовку данных, разработку архитектуры сети, ее обучение и последующее тестирование на новых данных. После успешного завершения этого процесса сверточную нейронную сеть можно интегрировать в различные приложения и системы для автоматического распознавания рукописных цифр в реальном времени, таких как мобильные приложения, веб-сервисы или десктопные программы, предназначенные для обработки и анализа изображений с цифрами, предоставленными пользователями.

Применение сверточных нейронных сетей для распознавания рукописных цифр имеет широкий спектр практических применений. Например, их можно использовать в банковской сфере для автоматического считывания рукописных чеков или документов, в почтовых системах для автоматической сортировки адресов, а также в приложениях для распознавания рукописного текста для упрощения ввода данных. Кроме того, сверточные нейронные сети могут быть интегрированы в системы безопасности для распознавания рукописных подписей или в системы медицинской диагностики для автоматического анализа рукописных записей врачей или результатов анализов. Таким образом, использование сверточных нейронных сетей для распознавания рукописных

цифр представляет собой мощный инструмент для автоматизации и улучшения эффективности различных процессов, требующих анализа рукописных данных.

2.1.3 Преимущества в распознавании рукописных цифр**:**

Структура с иерархией. Сверточные нейронные сети имеют иерархическую организацию, что позволяет им автоматически извлекать сложные иерархические характеристики из изображений. Это дает возможность сетям распознавать сложные шаблоны и структуры, что особенно важно при распознавании рукописных цифр, где каждая цифра имеет свою уникальную форму и структуру.

Общие параметры для различных областей изображения. Сверточные нейронные сети используют сверточные слои, где параметры общие для различных областей изображения. Это позволяет сетям эффективно использовать информацию из разных частей изображения и сократить количество обучаемых параметров. При распознавании рукописных цифр это позволяет сетям обобщить знания о форме и структуре цифр на различные области изображения.

Инвариантность к масштабированию и сдвигу. Сверточные нейронные сети обладают свойством инвариантности к изменениям размера и положения объектов на изображении. Это означает, что они способны распознавать цифры независимо от их размера и местоположения на изображении, что особенно важно для распознавания рукописных цифр.

Работа с разными стилями написания. Сверточные нейронные сети могут обучаться на различных стилях написания рукописных цифр. Благодаря их способности извлекать иерархические характеристики из данных, они успешно распознают цифры, написанные разными людьми.

Масштабируемость. Сверточные нейронные сети легко масштабируются для работы с разными объемами данных и задачами распознавания. Их можно обучать на небольших наборах данных для решения конкретных задач и масштабировать для работы с более крупными и разнообразными данными без

существенных изменений в архитектуре сети. Это делает их гибкими инструментами для распознавания рукописных цифр в различных сценариях.

2.1.4 Ограничения сверточных нейронных сетей в распознавании рукописных цифр:

Сверточные нейронные сети требуют большого объема размеченных данных для обучения. Для эффективного распознавания рукописных цифр, необходимо иметь достаточное количество размеченных примеров цифр, чтобы сеть могла научиться обобщать знания о форме и структуре цифр.

Сверточные нейронные сети могут быть чувствительны к вариациям в написании цифр. Например, если цифра написана с небольшими изменениями в форме или структуре, сеть может неправильно распознать ее. Это может быть проблемой при распознавании рукописных цифр, где каждый человек может писать цифры с некоторыми индивидуальными вариациями.

Сверточные нейронные сети могут быть вычислительно сложными и требовать больших вычислительных ресурсов для обучения и использования. Это может быть проблемой при использовании CNN на устройствах с ограниченными вычислительными возможностями, таких как мобильные устройства.

Для достижения оптимальной производительности сверточных нейронных сетей в распознавании рукописных цифр требуется тщательная настройка параметров модели, таких как количество слоев, размер ядер свертки, скорость обучения и другие. Неправильная настройка этих параметров может привести к ухудшению производительности сети или даже к ее неспособности к распознаванию цифр.

Сверточные нейронные сети могут быть чувствительны к шуму и искажениям на изображениях, что может привести к неправильному распознаванию. Например, наличие фонового шума или артефактов на изображениях может затруднить работу сети и уменьшить ее точность.

Ограниченная интерпретируемость. В связи с их сложной структурой и большим количеством параметров, сверточные нейронные сети могут быть менее интерпретируемыми по сравнению с более простыми моделями. Это может затруднить понимание того, как сеть принимает решения и какие признаки она использует для распознавания цифр, что усложняет диагностику и улучшение модели.

2.2 Vision Transformer (ViT)

Vision Transformer (ViT) представляет собой революционную архитектуру в области компьютерного зрения, которая применяет механизмы трансформеров, изначально разработанные для обработки последовательностей текста, к задачам распознавания изображений. Эта модель показала впечатляющие результаты, сравнимые и в некоторых случаях превосходящие традиционные сверточные нейронные сети (CNN). ViT предназначен для обработки последовательностей данных и использует механизм внимания (attention) для определения важных частей входной последовательности при генерации выходной. Трансформеры значительно улучшили результаты в обработке естественного языка (NLP), таких как машинный перевод, суммаризация текста и вопросно-ответные системы.

Применение к изображениям

Основная идея ViT заключается в том, чтобы применять архитектуру трансформера к изображениям. В традиционных CNN, фильтры свертки сканируют изображения для извлечения локальных признаков. ViT разбивает изображение на небольшие участки (patches) и обрабатывает их как последовательность токенов, аналогично словам в предложении в NLP.

Разбиение на патчи. Разбиение изображения на небольшие участки фиксированного размера (обычно 16x16 пикселей). Каждому участку назначается позиционный эмбеддинг, чтобы сохранить информацию о его расположении в исходном изображении. Эти патчи затем линейно проецируются в векторы фиксированной длины и подаются на вход трансформера.

2.2.1 Архитектура ViT

Архитектура ViT состоит из нескольких слоев трансформера. Каждый слой включает в себя механизмы само-внимания (self-attention) и полностью связанные слои (feed-forward layers). В механизме само-внимания каждая часть изображения может взаимодействовать с любой другой частью, что позволяет модели учитывать глобальные зависимости.

Основные компоненты трансформера включают:

Self-Attention: Вычисляет внимание (важность) каждого патча относительно других.

Multi-Head Attention: Использует несколько голов внимания для захвата различных аспектов взаимосвязей между патчами.

Feed-Forward Network: Применяет нелинейные преобразования к каждому патчу.

Layer Normalization: Нормализует входные данные слоя, что способствует стабильности и ускорению обучения.

Residual Connections: Сохраняет информацию из предыдущих слоев, облегчая обучение глубоких сетей.

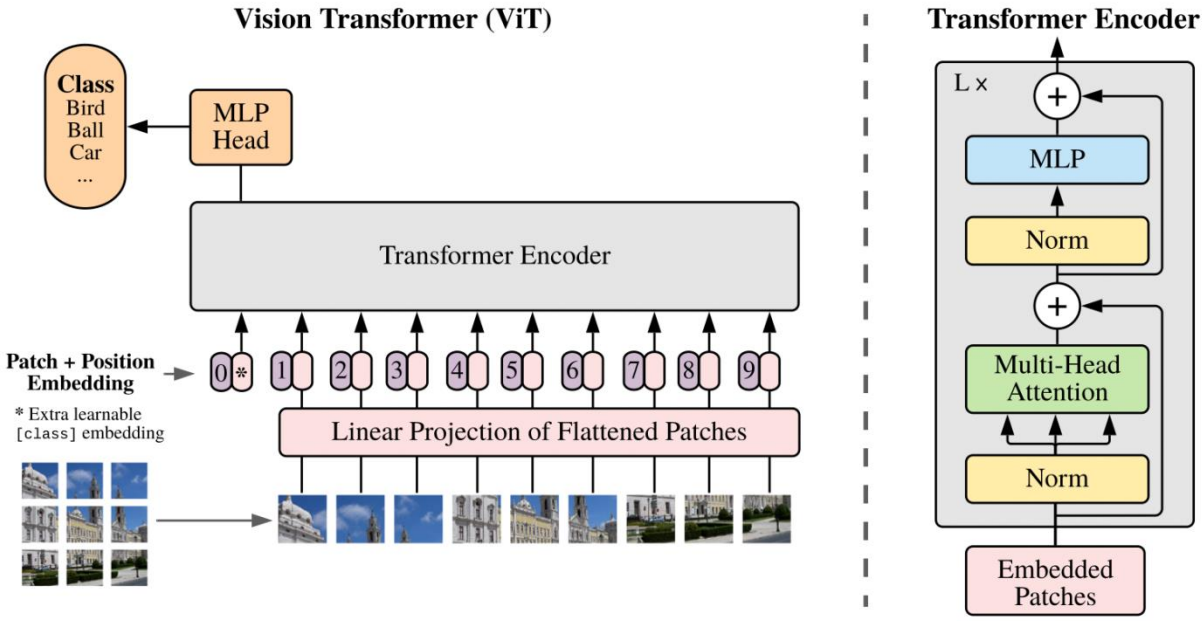


Рисунок 1 – Архитектура трансформера

2.2.2 Преимущества ViT

Глобальное внимание: ViT может захватывать долгосрочные зависимости между различными частями изображения благодаря механизму внимания, что затруднительно для традиционных CNN.

Масштабируемость: ViT показывает лучшие результаты по мере увеличения размеров моделей и наборов данных. Большие модели ViT, обученные на больших наборах данных, значительно превосходят традиционные методы.

Универсальность: ViT можно применять к различным задачам компьютерного зрения, включая классификацию, сегментацию и обнаружение объектов.

2.2.3 Обучение ViT

ViT требует больших наборов данных для достижения высоких результатов. В оригинальной статье авторы использовали набор данных JFT-300M (300 миллионов изображений) для предобучения модели. Модель ViT затем может быть дообучена на меньших наборах данных (например, ImageNet) для конкретных задач.

ViT достиг впечатляющих результатов на популярных бенчмарках, к примеру в ImageNet. Исследования также показывают, что ViT требует меньше вычислительных ресурсов для предсказаний по сравнению с CNN аналогичной точности.

Vision Transformer представляет собой значительный шаг вперед в области компьютерного зрения. Применяя мощные механизмы трансформеров к обработке изображений, ViT демонстрирует, что традиционные подходы к архитектуре нейронных сетей могут быть переосмыслены и адаптированы для достижения лучших результатов. Будущее исследований ViT включает в себя оптимизацию обучения на меньших наборах данных, исследование гибридных архитектур (сочетание трансформеров и сверток) и применение ViT к более широкому спектру задач компьютерного зрения.

3 Разработка программного обеспечения

3.1 Выбор языка программирования

Для разработки программного обеспечения используются языки программирования и под каждое выбирается язык который по функционалу будет выполнять все требуемые от него задачи. Выбор языка должен строится исходя из архитектуры приложения, функционала который мы хотим получить, масштабируемости и т.п.

Для решения задач с применением искусственного интелекта выделяется группа языков которая позволит получить желаемый результат и обеспечит быструю скорость выполнения.

- Python;
- Java;
- C++;
- Matlab.

Для реализации системы распознавания текста был выбран язык Python, так как он выделяется среди остальных языков программирования тем, что на нем

достаточно быстро вести разработку, прост в освоении, имеет большое количество библиотек для работы с искусственным интелектом и нейросетями, и поддерживается практически на любой платформе.

Код Python поддерживается на большинстве современных операционных систем, сам код кроссплатформенный, что позволяет использовать программу на различных устройствах, не деля программу под какую-то конкретную систему.

Немаловажным фактором для выбора является скорость выполнения кода, а такая скорость обеспечена тем, что основные Python библиотеки написаны на C++, который быстрее чем другие высокоуровневые языки программирования.

3.2 Используемые библиотеки

Также Библиотеки, которые были использованы для разработки программы:

- keras (2.0.2)
- TensoFlow (2.0.1)
- TensoFlow-gpu (1.2.1)
- Matplotlib (3.1)
- PyQt5 (5.9)
- Набор данных MNIST

Keras - это высокоуровневый API нейронной сети, ориентированный на удобство использования, быстрое создание прототипов, модульность и расширяемость. Набор данных MNIST в большой базе рукописных цифр (от 0 до 9). Он содержит 60000 обучающих изображений и 10000 тестовых изображений каждое размером 28x28.

GUI (графический интерфейс пользователя) предоставляет интерфейс для взаимодействия человека с машиной. В этой части я использовал библиотеку PyQt5 для создания графического интерфейса.

PyQt5 - это набор Python библиотек для создания графического интерфейса на базе платформы Qt5 от компании Digia. Библиотека Qt является одной из

самых мощных библиотек GUI (графического интерфейса пользователя), которая может быть использована для быстрого построения интерфейса, включающий в себя несколько основных элементов, таких как метки, кнопки, текстовые поля и т. д. И эти модули имеют десятки характеристик, таких как размер, цвет, шрифт, рельеф, растровое изображение, курсор для их настройки.

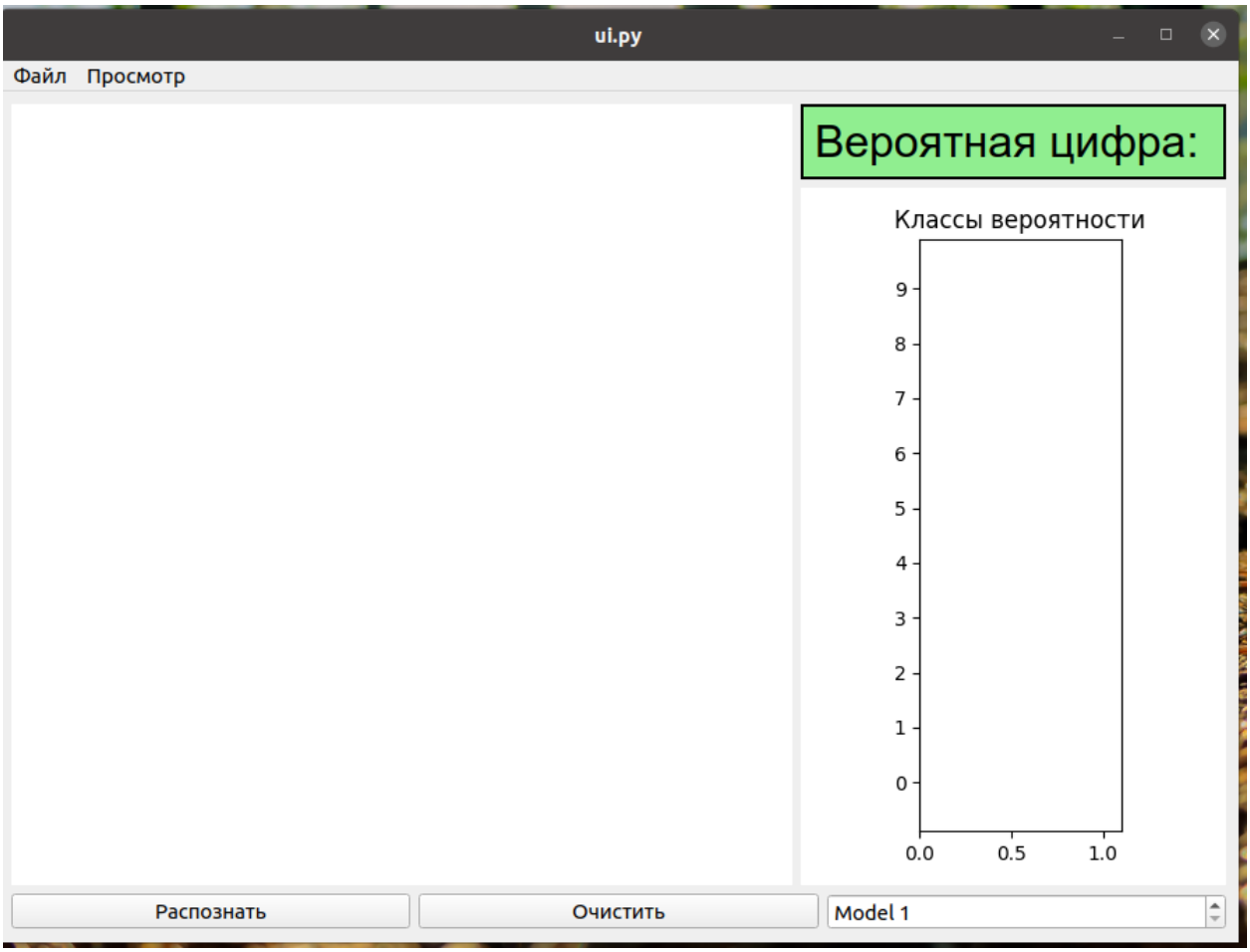


Рисунок 2 – Графический интерфейс

Сверточная нейронная сеть была использована для реализации предложенной модели. Эта модель была реализована с использованием высокоуровневой нейронной сети API Keras.

3.3 Описание работы итоговой нейронной сети и замеры точности

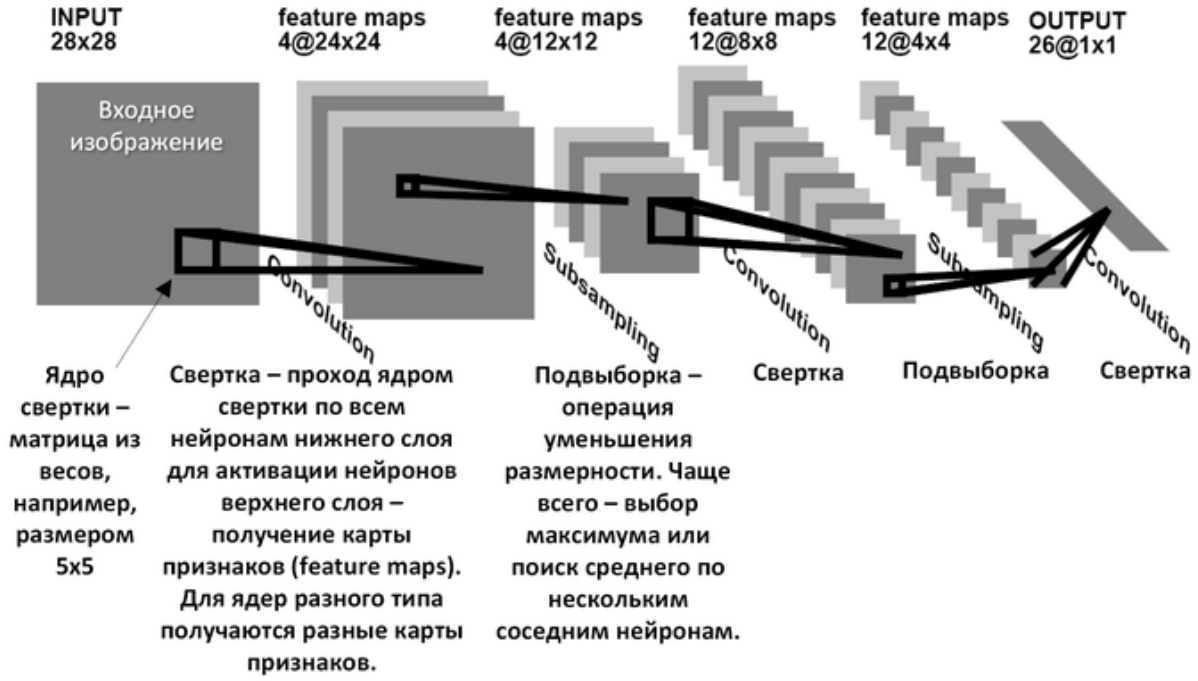


Рисунок 3 – Архитекура сверточной нейронной сети

Сверточная нейронная сеть представляет собой чередование сверточных слоев (convolution layers), субдискретизирующих слоев (subsampling layers) и при наличии полносвязных слоев (fully-connected layer) на выходе. Все три вида слоев могут чередоваться в произвольном порядке. В сверточном слое нейроны, которые используют одни и те же веса, объединяются в карты признаков (feature maps), а каждый нейрон карты признаков связан с частью нейронов предыдущего слоя. При вычислении сети получается, что каждый нейрон выполняет свертку некоторой области предыдущего слоя (определяемой множеством нейронов, связанных с данным нейроном).

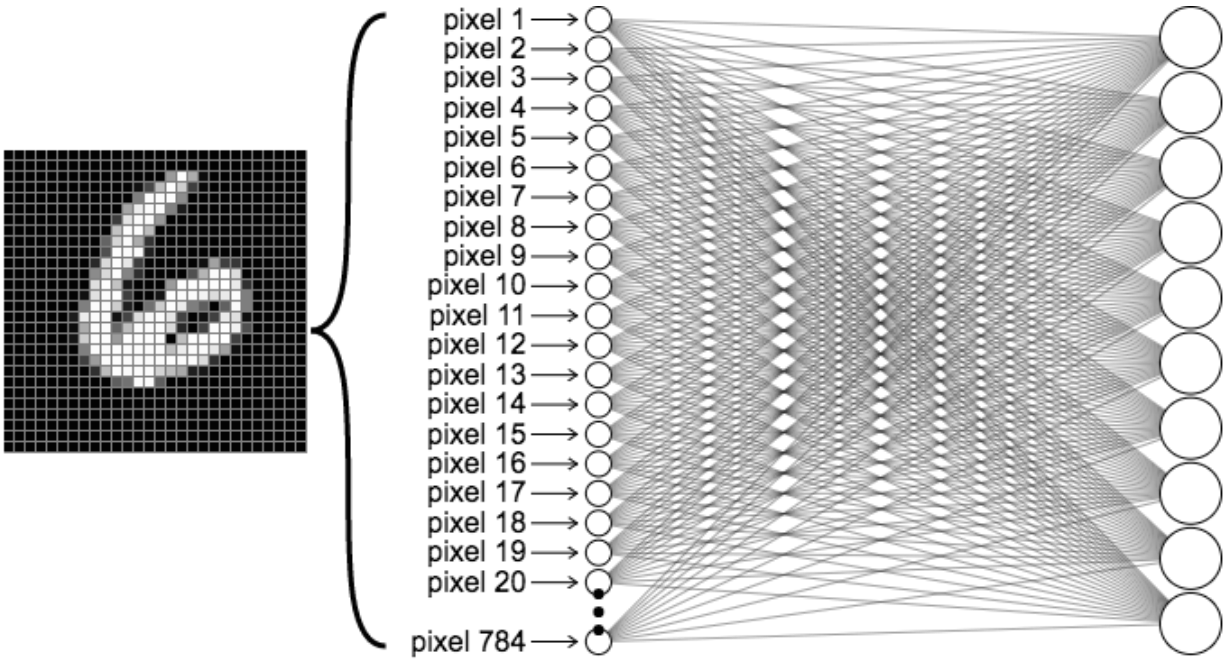


Рисунок 4 – Процесс распознования цифры

Название архитектура сети получила из-за наличия операции свёртки, суть которой в том, что каждый фрагмент изображения умножается на матрицу (ядро) свёртки поэлементно, а результат суммируется и записывается в аналогичную позицию выходного изображения.

Нейронные сети получают входные данные (один вектор), после чего трансформируют информацию, проводя ее через ряд скрытых слоев. Каждый скрытый слой состоит из множества нейронов, где всякий нейрон имеет устойчивую связь со всеми нейронами в предыдущем слое и где нейроны в функции одного слоя полностью независимы друг от друга и не имеют общих соединений. Последний полносвязный слой называется выходным слоем, и в настройках классификации он демонстрирует число классов.

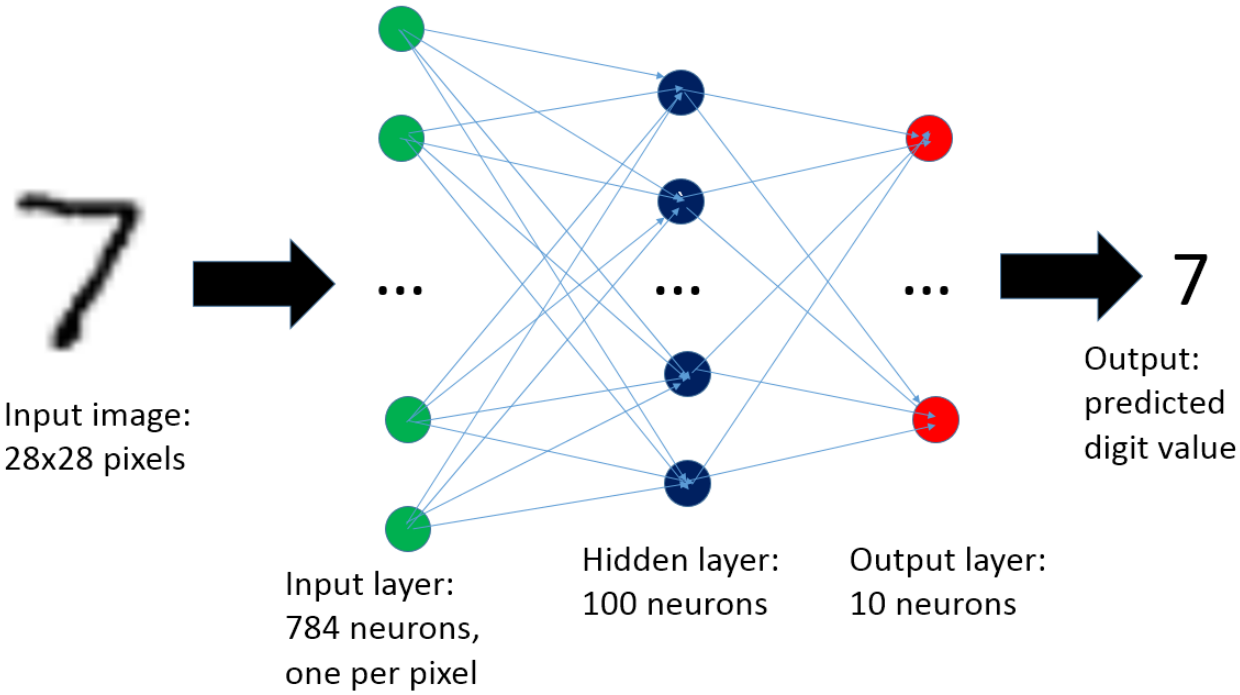


Рисунок 5 – Распознование с использованием перцептрона

В свёрточной нейронной сети в операции свёртки используется лишь ограниченная матрица весов небольшого размера, которую «двигают» по всему обрабатываемому слою (в самом начале — непосредственно по входному изображению), формируя после каждого сдвига сигнал активации для нейрона следующего слоя с аналогичной позицией. То есть для различных нейронов выходного слоя используются одна и та же матрица весов, которую также называют ядром свёртки. Её интерпретируют как графическое кодирование какого-либо признака, например, наличие наклонной линии под определённым углом. Тогда следующий слой, получившийся в результате операции свёртки такой матрицей весов, показывает наличие данного признака в обрабатываемом слое и её координаты, формируя так называемую карту признаков (англ. feature

map).

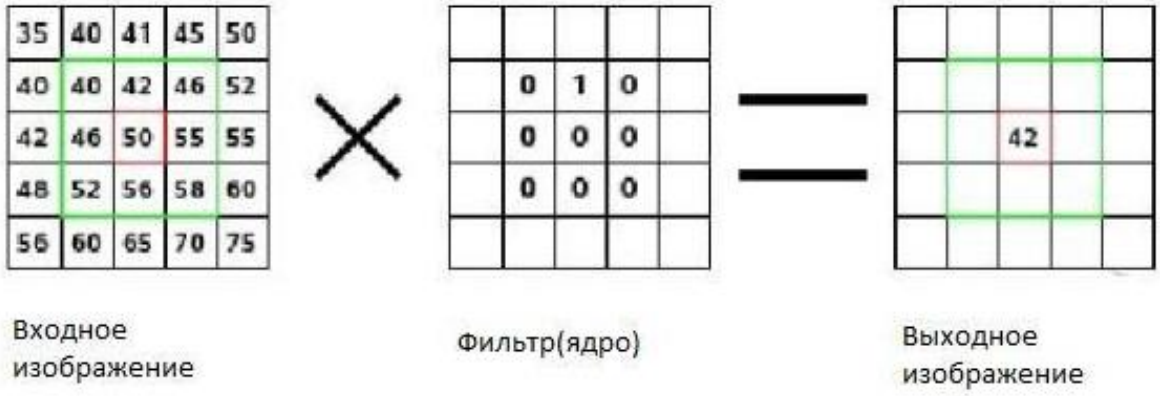


Рисунок 6 – Карта признаков

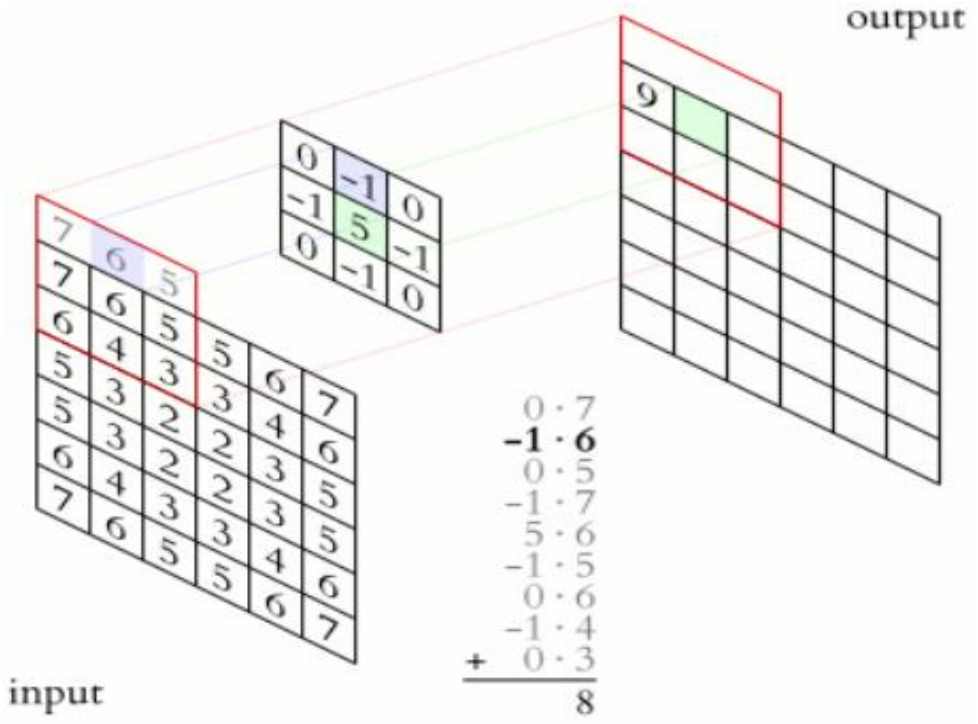


Рисунок 7 – Обработка карты признаков

В сверточной нейронной сети используется набор весов, а не только один, чтобы кодировать различные элементы изображения, такие как линии и дуги под разными углами. Эти веса формируются в процессе обучения сети с помощью классического метода обратного распространения ошибки, а не задаются заранее исследователем. Каждый набор весов формирует собственную карту признаков, делая нейронную сеть многоканальной, то есть с множеством независимых карт признаков на одном слое. При переборе слоя матрицей весов её обычно

передвигают на небольшое расстояние, а не на полный шаг, чтобы не пропустить искомый признак.

Рассмотрим типичную структуру сверточной нейронной сети:

Слой свертки - это основной строительный блок сверточной нейронной сети. В этом слое для каждого канала присутствует свой фильтр, и ядро свертки обрабатывает предыдущий слой по фрагментам. Весовые коэффициенты ядра свертки устанавливаются в процессе обучения и неизвестны заранее.

Каждый нейрон сверточного слоя соединен только с ограниченным числом нейронов предыдущего слоя, что делает сверточный слой аналогом операции свертки с матрицей весов небольшого размера (ядром свертки), которую "двигают" по всему обрабатываемому слою. Еще одной особенностью сверточного слоя является небольшое уменьшение размера изображения за счет краевых эффектов.

Особенностью сверточного слоя является сравнительно небольшое количество параметров, устанавливаемое при обучении. Например, если размер исходного изображения составляет 100x100 пикселей с тремя каналами, а сверточный слой использует фильтры с ядром 3x3 пикселя и выходом на 6 каналов, то в процессе обучения определяется только 9 весов ядра. В данном случае слой требует нахождения всего 162 параметров, что значительно меньше, чем количество параметров в полносвязной нейронной сети.

Имея двумерное изображение I и небольшую матрицу K размерности h* ω (ядро свертки), построенная таким образом, что графически кодирует какой-либо признак, мы вычисляем свернутое изображение I * K, накладывая ядро на изображение всеми возможными способами и записывая сумму произведений элементов исходного изображения и ядра:

$$(I * K)_{xy} = \sum_{i=1}^{h} \sum_{j=1}^{w} K_{ij} \times I_{x+i-1,y+j-1}$$

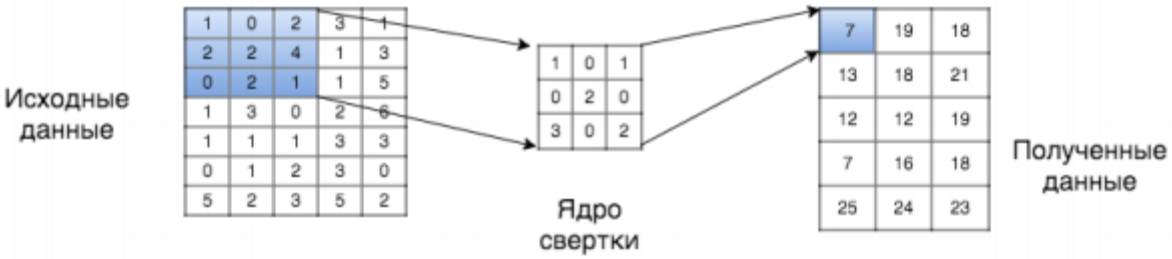


Рисунок 8 – Сверточный слой с ядром свертки

Скалярный результат каждой свёртки попадает на функцию активации, которая представляет собой некую нелинейную функцию. Слой активации обычно логически объединяют со слоем свёртки (считают, что функция активации встроена в слой свёртки). Функция нелинейности может быть любой по выбору исследователя, традиционно для этого использовали функции типа гиперболического тангенса:

$$\operatorname{th} x = \frac{\operatorname{sh} x}{\operatorname{ch} x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$

или сигмоиды:

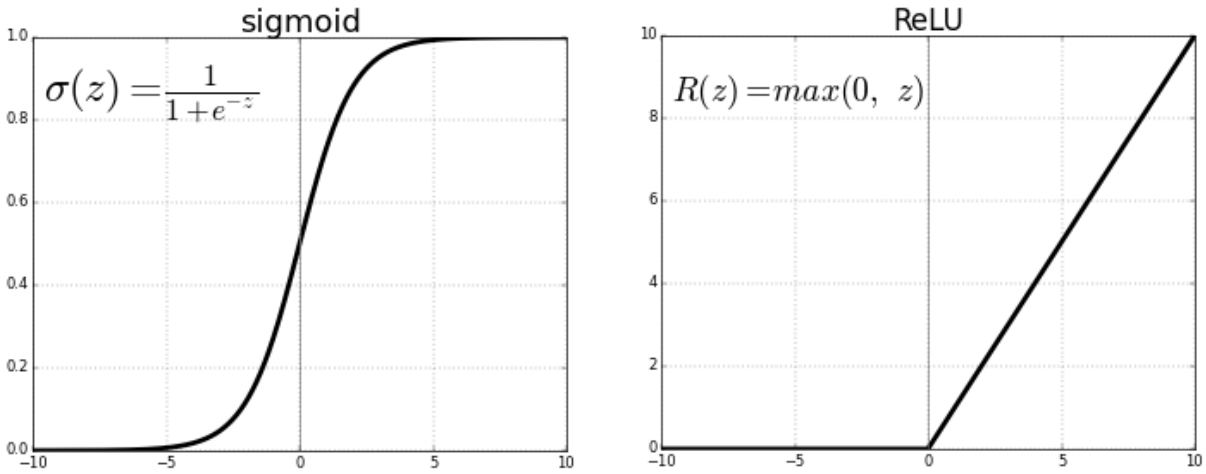


Рисунок 9 - Сигмоид

Однако в 2000х годах была предложена и исследована новая функция активации — ReLU (сокращение от англ. rectified linear unit), которая позволила существенно ускорить процесс обучения и одновременно упростить вычисления (за счет простоты самой функции), что означает блок линейной ректификации.

То есть по сути это операция отсечения отрицательной части скалярной величины. Эта функция и её модификации (Noisy ReLU, Leaky ReLU и другие) являются наиболее часто используемыми функциями активации в глубоких нейросетях, в частности, в свёрточных.

Слой пулинга (подвыборки, субдискретизации) представляет собой нелинейное уплотнение карты признаков, при этом группа пикселей (обычно размера 2×2) уплотняется до одного пикселя, проходя нелинейное преобразование. Наиболее употребительна при этом функция максимума.

MAX-pooling слой:

$$\widehat{c} = max\{c\}$$

Преобразования затрагивают непересекающиеся прямоугольники или квадраты, каждый из которых ужимается в один пиксель, при этом выбирается пиксель, имеющий максимальное значение. Операция пулинга позволяет существенно уменьшить пространственный объём изображения.

Если на предыдущей операции свёртки уже были выявлены некоторые признаки, то для дальнейшей обработки настолько подробное изображение уже не нужно, и оно уплотняется до менее подробного. К тому же фильтрация уже ненужных деталей помогает не переобучаться. Слой пулинга, как правило, вставляется после слоя свёртки перед слоем следующей свёртки.

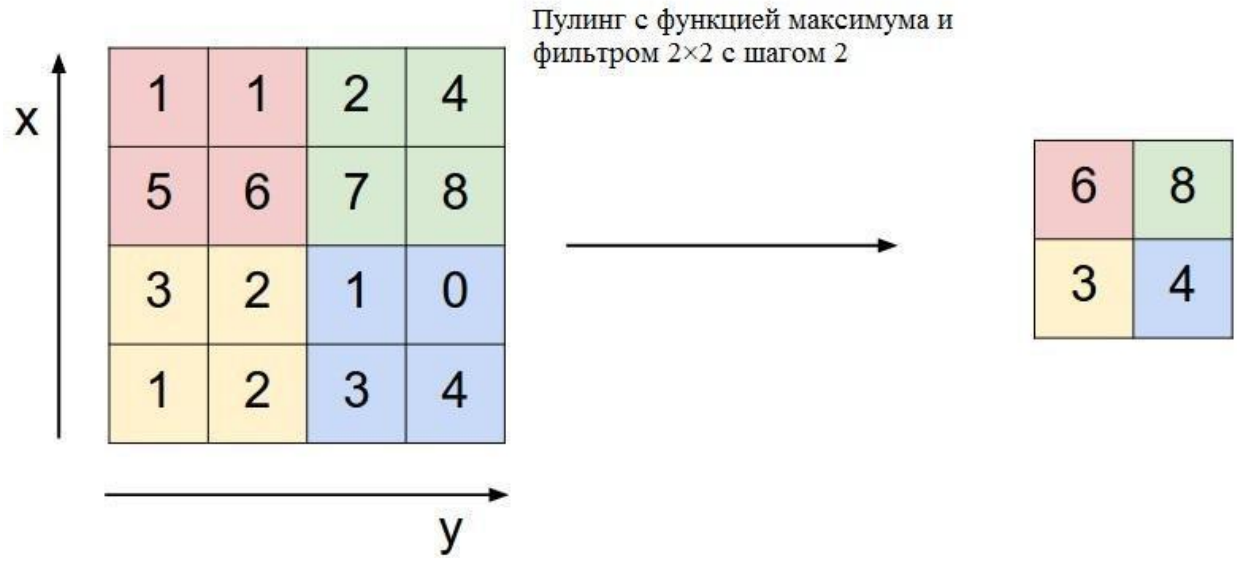


Рисунок 10 - Пулинг

Слой состоит из определенного количества ядер K (с аддитивными составляющими смещения для каждого ядра) и вычисляет свертку выходного изображения предыдущего слоя с помощью каждого из ядер, каждый раз прибавляя составляющую смещения. В конце концов ко всему выходному изображению может быть применена функция активации σ. Обычно входной поток для сверточного слоя состоит из d каналов, например, red/green/blue для входного слоя, и в этом случае ядра тоже расширяют таким образом, чтобы они также состояли из d каналов; получается следующая формула для одного канала выходного изображения сверточного слоя, где K — ядро, а b — составляющая смещения:

Сверточный слой

$$(I, K)_{x,y} = \sigma(b + \sum_{i=1}^{h} \sum_{j=1}^{w} \sum_{k=1}^{d} K_{ijk} \times I_{x+i-1,y+j-1,k})$$

Dropout слой (dropout регуляризация) способ борьбы с переобучением в нейронных сетях, обучение которых обычно производят стохастическим градиентным спуском, случайно выбирая некоторые объекты из выборки. Dropout регуляризация заключается в изменение структуры сети: каждый нейрон выбрасывается с некоторой вероятностью p. По такой прореженной сети

производится обучение, для оставшихся весов делается градиентный шаг, после чего все выброшенные нейроны возвращаются в нейросеть.

Таким образом, на каждом шаге стохастического градиента мы настраиваем одну из возможных 2^N архитектур сети, где под архитектурой мы понимаем структуру связей между нейронами, а через N обозначаем суммарное число нейронов. При тестировании нейросети нейроны уже не выбрасываются, но выход каждого нейрона умножается на (1 − p) — благодаря этому на выходе нейрона мы будем получать математическое ожидание его ответа по всем 2^N архитектурам. Таким образом, обученную с помощью dropout-регуляризации нейросеть можно рассматривать как результат усреднения 2^N сетей.

Предложенная модель для распознавания рукописных цифр (0-9), реализованная с использованием CNN с помощью высокоуровневого нейросетевого API Keras, который работает на вершине тензорного потока. Конфигурации, используемые для модели CNN, следующие:

Размер изображения: 28x28, 2 слоя свертки, Фильтр: 3x3x1, Функция активации: Relu

Функция Relu: f (x) = max (0, x). Цель ReLU - представить нелинейность в нашей ConvNet; Размер бассейна: 2x2; 2 полностью подключенных слоя/

Разработанная модель была запущена и протестирована с выборкой размером 1000. Следующая таблица показывает полученный результат.

Потери	Точность	Количество эпох
0.29	91.1	2
0.19	94.1	3
0.15	95.3	4
0.086	97.3	5
0.049	98.4	6

Из результата видно, что точность обычно увеличивается, а потери уменьшаются с увеличением числа эпох.

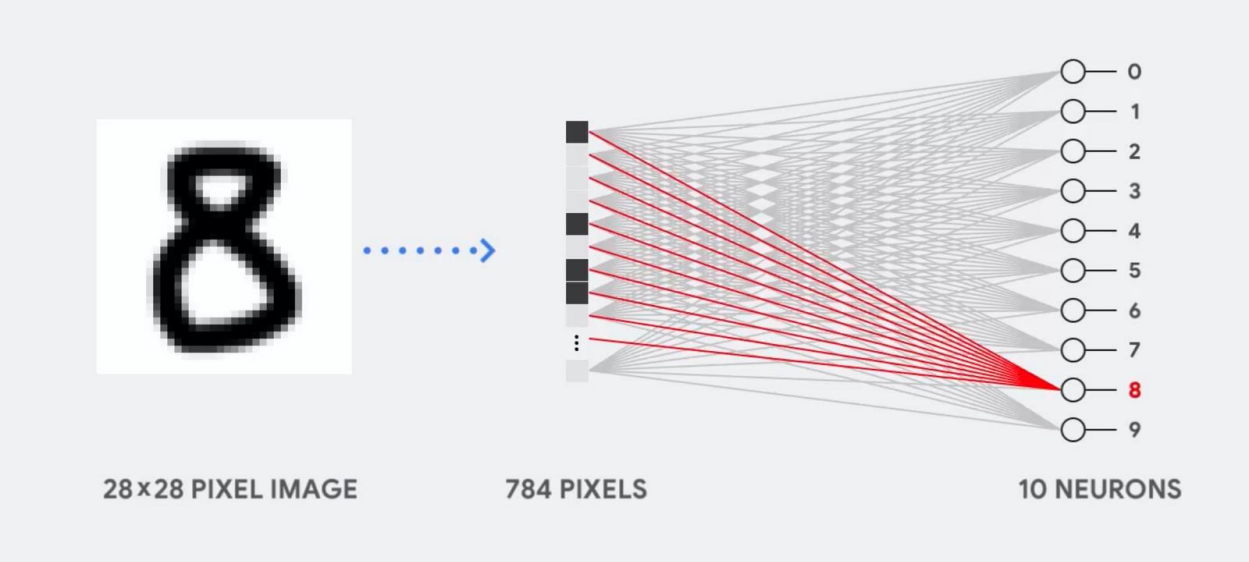


Рисунок 11 – Многослойный перцептрон

Если система классификации была обучена различать цифры, матрица ошибок будет суммировать результаты тестирования алгоритма для дальнейшей проверки. Предполагая выборку из 1000 цифр, полученная матрица ошибок будет выглядеть следующим образом:

	0	1	2	3	4	5	6	7	8	9
0	95	0	0	0	0	0	0	0	2	0
1	0	97	0	0	0	0	0	1	0	0
2	0	0	89	0	0	0	0	0	0	0
3	0	0	1	103	0	0	0	0	1	0
4	0	0	0	0	100	0	0	0	0	0
5	0	0	0	1	0	99	1	0	0	0
6	1	0	0	0	0	1	98	0	0	0
7	0	3	0	0	0	0	0	96	0	0
8	0	0	0	1	0	0	0	0	104	0
9	0	0	0	0	0	1	0	0	0	101

Все правильно детектированные числа расположены на диагонали таблицы, поэтому визуально проверить таблицу на наличие ошибок распознавания легко, поскольку они будут представлены значениями вне диагонали.

В зависимости от решаемых задач, пользу от ускорения можно выразить как в сокращении затраченного времени на вычисления, так и в увеличении потенциально допустимого объёма обрабатываемых данных. Ниже на диаграммах представлена зависимость времени обучение нейронной сети от количества нейронов в условиях большого количества данных.

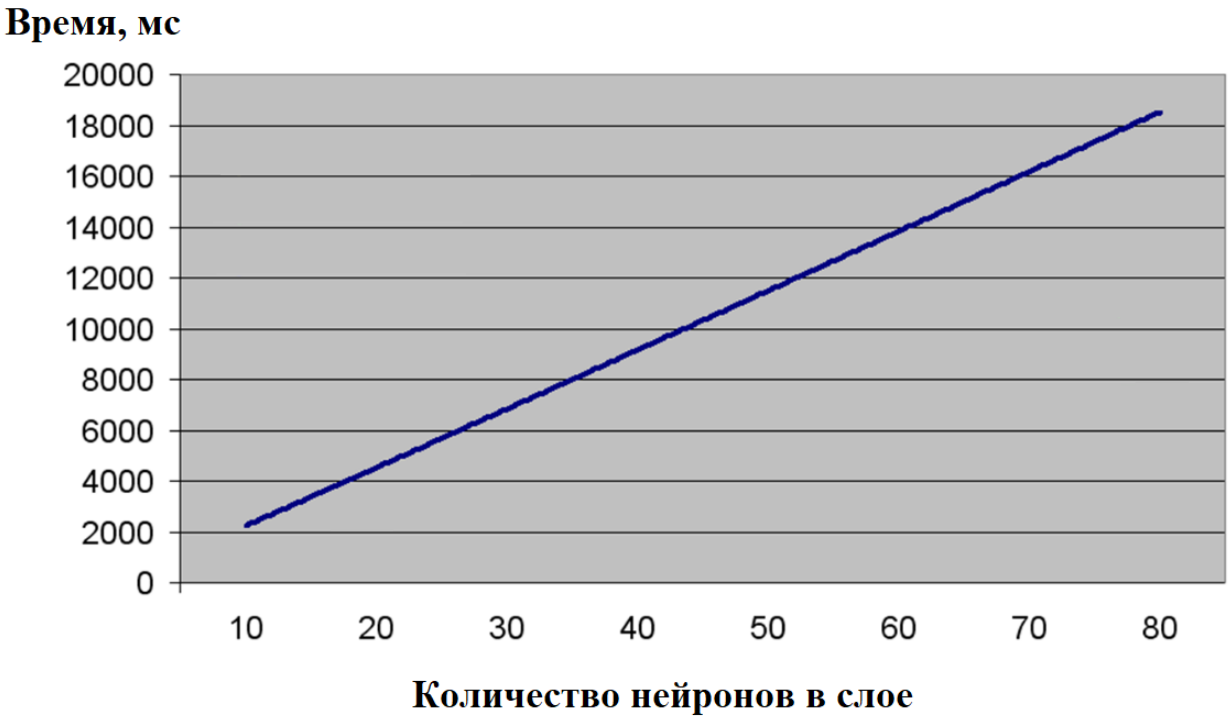


Рисунок 12- График зависимости времени от нейронов 1

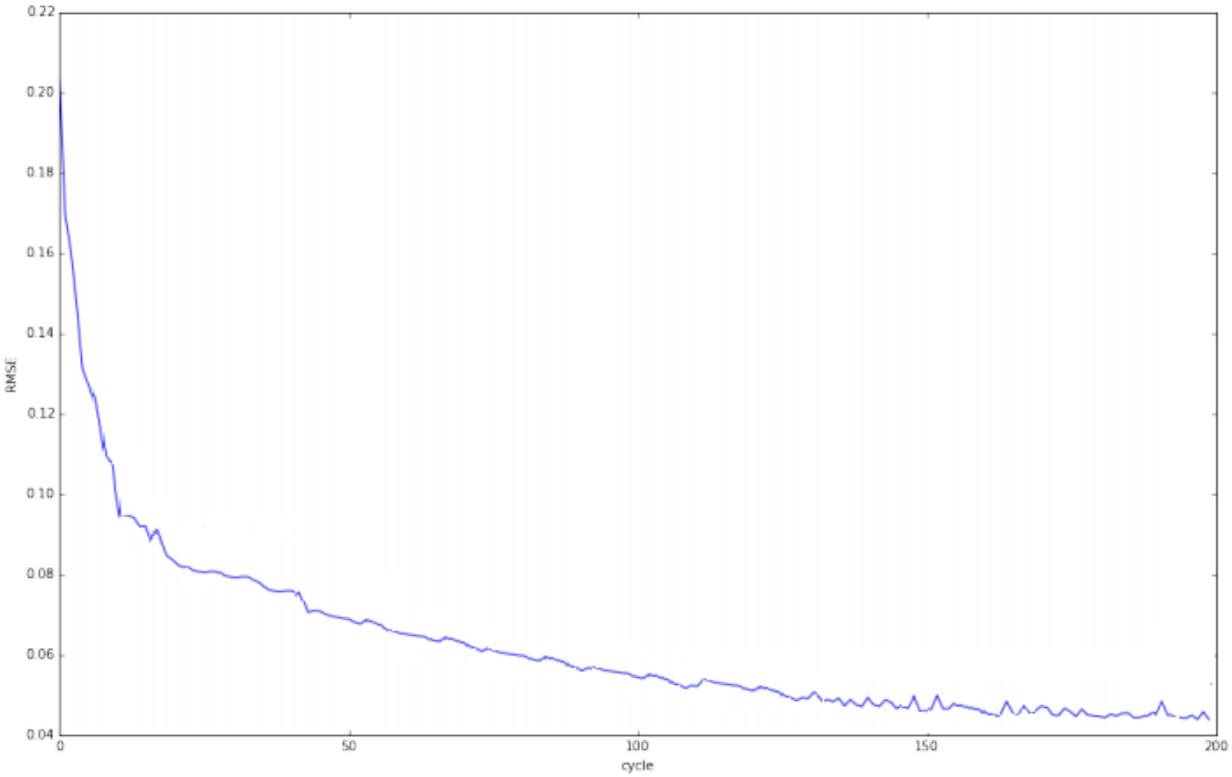


Рисунок 13 - График зависимости времени от нейронов 2

Среднеквадратичная ошибка

$$RMSE = \sqrt{\frac{\sum_{t=1}^{n}(y_t - \widetilde{y}_t)^2}{n}},$$

где y_t - фактическое значение, $\widetilde{y}_t$ - спрогнозированное значение.

Для оценки результатов распознавания используется средняя абсолютная ошибка в процентах (MAPE):

$$MAPE = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{y_t - \widetilde{y}_t}{y_t}\right|$$

Величина MAPE на тестовом множестве составила ≈ 2.47%, что в целом показывает корректность полученной модели.

Исходя из полученных результатов и оценки построенной модели, можно сделать вывод, что разработанная сверточная нейронные сеть, обученная на подмножестве большего набора данных из базы NIST, являются хорошим

инструментом для распознавания рукописных чисел, и их использование в этой области вполне оправдывает себя.

Модель обучена на наборе данных MNIST, которая содержит обучающий набор из 60000 примеров и набор тестов из 10000 примеров.

В обучающую модель сначала я загружаю обучающие наборы из интернета и использую алгоритм keras, библиотеку tensorflow и numpy, чтобы узнать о наборах и получить окончательную модель, которая в общей сложности составляет 60 000 образцов. Итоговый результат обучения показал, что частота ошибок составляет 0,80%, и этот показатель показан ниже на рисунке.

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
 - 173s - loss: 0.3881 - acc: 0.8819 - val_loss: 0.0880 - val_acc: 0.9722
Epoch 2/10
 - 162s - loss: 0.0989 - acc: 0.9700 - val_loss: 0.0561 - val_acc: 0.9822
Epoch 3/10
 - 146s - loss: 0.0741 - acc: 0.9773 - val_loss: 0.0414 - val_acc: 0.9864
Epoch 4/10
 - 130s - loss: 0.0598 - acc: 0.9817 - val_loss: 0.0355 - val_acc: 0.9874
Epoch 5/10
 - 130s - loss: 0.0518 - acc: 0.9836 - val_loss: 0.0349 - val_acc: 0.9889
Epoch 6/10
 - 128s - loss: 0.0445 - acc: 0.9857 - val_loss: 0.0291 - val_acc: 0.9900
Epoch 7/10
 - 131s - loss: 0.0387 - acc: 0.9880 - val_loss: 0.0294 - val_acc: 0.9904
Epoch 8/10
 - 140s - loss: 0.0344 - acc: 0.9892 - val_loss: 0.0277 - val_acc: 0.9903
Epoch 9/10
 - 143s - loss: 0.0318 - acc: 0.9902 - val_loss: 0.0240 - val_acc: 0.9922
Epoch 10/10
 - 146s - loss: 0.0303 - acc: 0.9903 - val_loss: 0.0269 - val_acc: 0.9920
Baseline Error: 0.80%
```

Рисунок 14 - Окончательный результат обучения

Если быть точным и проводить тестирование в глобальных масштабах, то эта основанная на CNN модель для распознавания рукописных цифр достигает точности подтверждения 92,2% после обучения в течение 14 эпох.

В основе трансформера лежат механизмы самовнимания (self-attention) и полностью соединенные слои, что позволяет эффективно обрабатывать последовательные данные.

Основные компоненты архитектуры трансформера: входные эмбеддинги. Токенизация - входные данные (например, текст) разбиваются на токены, эмбеддинг токенов - каждый токен преобразуется в эмбеддинг - вектор фиксированной размерности и позиционные эмбеддинги, они добавляются к эмбеддингам токенов, чтобы учитывать позиционную информацию в последовательности.

Энкодер. N слоев энкодера - трансформер использует многослойный энкодер, состоящий из нескольких одинаковых слоев.

Слои энкодера:

Механизм самовнимания (Self-Attention Mechanism): Позволяет каждому токену обращать внимание на все остальные токены в последовательности, взвешивая их по значимости и многоголовое внимание (Multi-Head Attention) которое разделяет векторы на несколько частей (голов), применяется механизм внимания.

Мы рассмотрим scaled dot-product attention (механизм на основе скалярного произведения), поскольку сегодня это самый популярный способ.

Наша цель — создать контекстуализированные эмбеддинги с помощью линейных комбинаций исходных эмбеддингов. Вся необходимая информация уже закодирована в наших обученных векторах, и теперь нам нужно лишь вычислить веса. Для этого необходимо определить, какие токены взаимно релевантны. Введем понятие сходства между двумя эмбеддингами, которое можно измерить с помощью скалярного произведения. Наша задача — обучить такие эмбеддинги, чтобы увеличение значения скалярного произведения

соответствовало увеличению сходства между изображениями

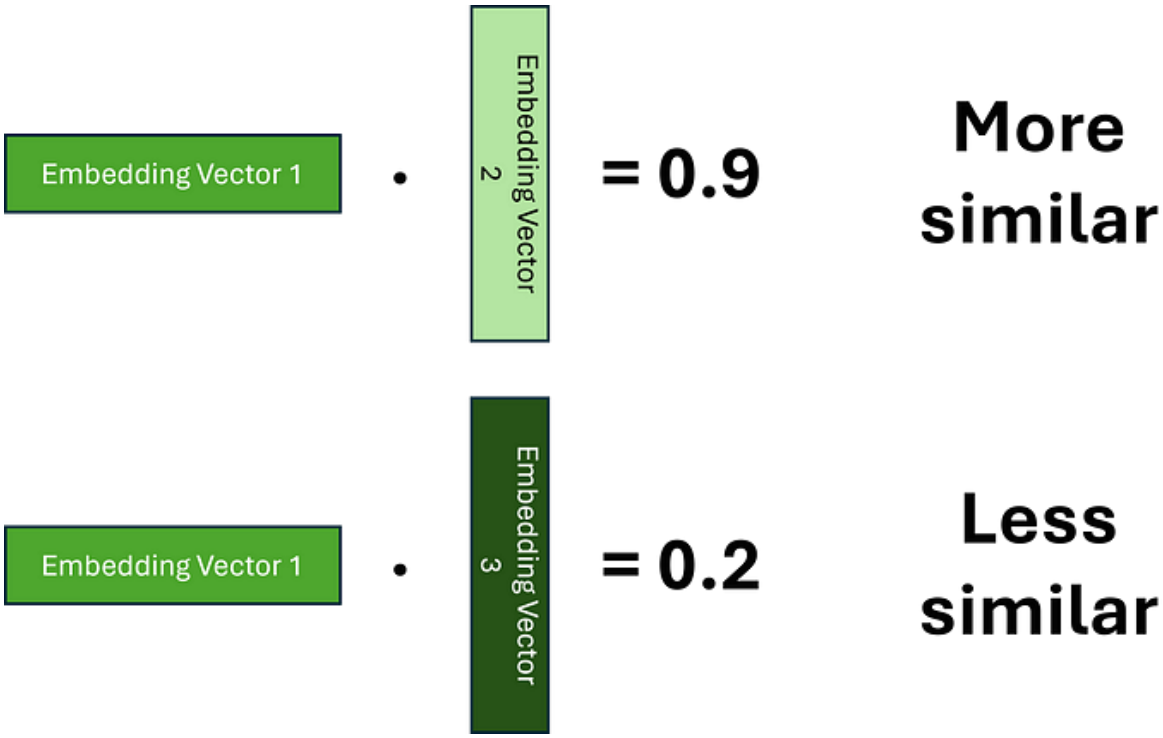


Рисунок 15 – Входные эмбеддинги

Так как нужно вычислить взаимосвязь между токенами, требуется получить матрицу весов (attention scores), которую называют оценкой внимания. Для проверки того что сумма веов равняется единице, нужно применить многопеременную логистическую функцию (МЛФ). Но перед ее применением умножим оценки внимания на поправочный коэффициент. Если предварительно этого не сделать то МЛФ будет возвращать маленькие градиенты для больших оценок внимания. А это может привести к проблеме исчезающего градиента во время обучения.

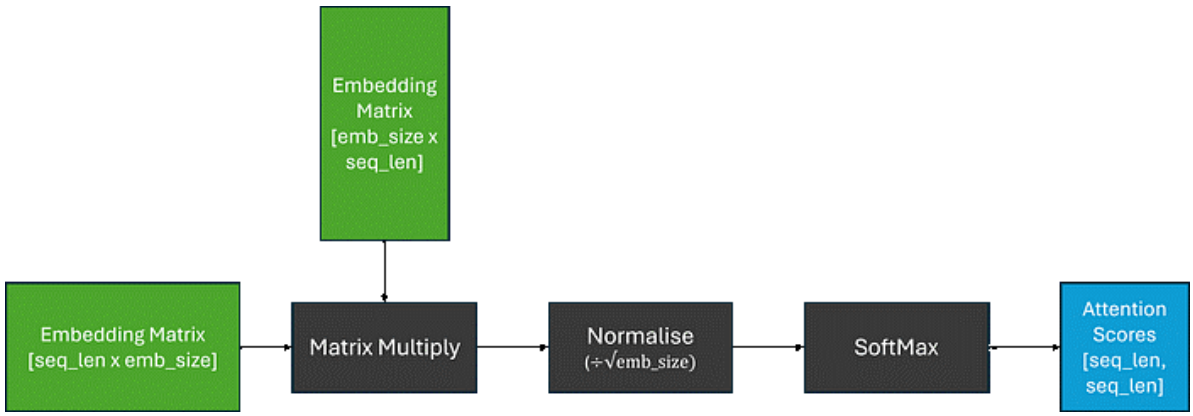


Рисунок 16 – Блок-схема применения эмбеддингов и нормализации

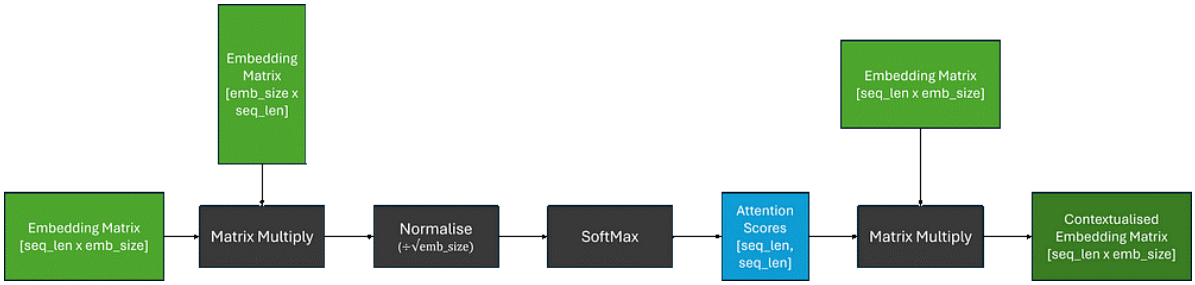


Рисунок 17 – Блок-схема вычисления внимания

Упрощенное вычисление внимания. Предполагается, что эмбеддинги позиционно закодированы.

Хотя модель может выучить достаточно сложные эмбеддинги для синтеза оценки внимания и контекстуальных эмбеддингов, мы пытаемся уместить в них слишком много информации. Поэтому для облегчения обучения модели вместо прямого использования матрицы эмбеддингов будем передавать её через три независимых линейных слоя (матричных умножений). Это позволит модели сосредоточиться на разных частях эмбеддингов:

Разработанная программа предоставляет удобный пользовательский интерфейс для рисования цифр и автоматического их распознавания.

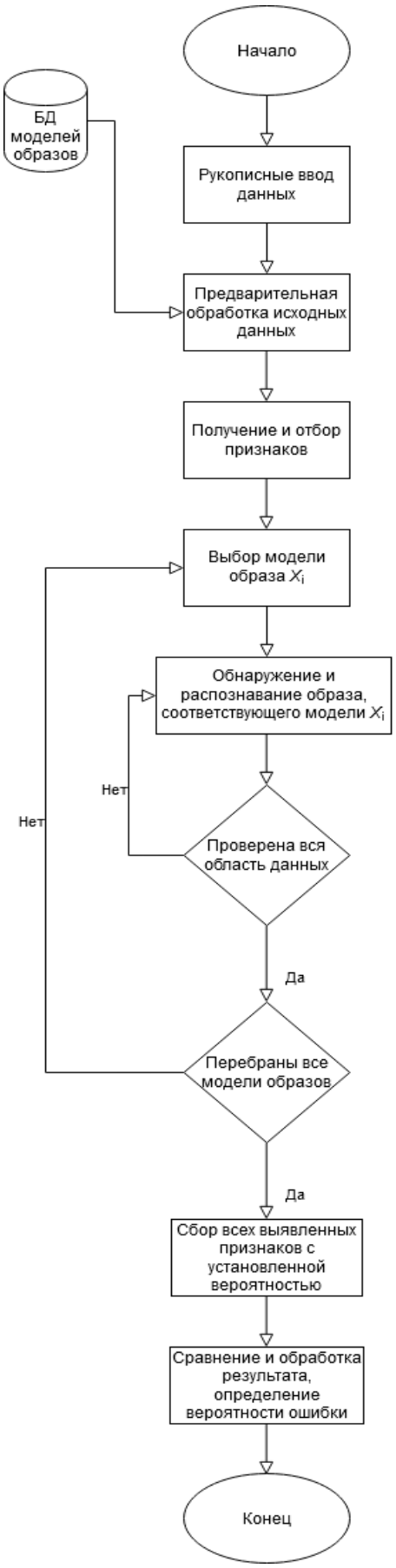
При запуске приложения пользователь видит интерфейс, включающий виджет для рисования, кнопку для очистки планшета и текстовое поле для вывода результатов распознавания. Приложение использует графический интерфейс для обеспечения удобства взаимодействия с пользователем.

Как только пользователь нарисовал цифру на планшете, он может нажать кнопку "Распознать", и программа начнет процесс анализа изображения. Для этого программа делает скриншот области планшета с нарисованной цифрой и передает его на вход нейронной сети.

Полученное изображение масштабируется до размера 28x28 пикселей, который соответствует ожидаемому формату для входных данных нейронной сети. Затем каждый пиксель изображения анализируется: если его интенсивность не является белой, то он интерпретируется как часть цифры и преобразуется в единицу, иначе — в ноль. Полученный массив пикселей подается на вход нейронной сети.

Нейронная сеть, предварительно загруженная из файла, производит распознавание цифры. Она обучена на большом наборе размеченных данных, что позволяет ей эффективно определять, какая цифра изображена на входном изображении. Результат предсказания отображается в текстовом поле пользовательского интерфейса.

Таким образом, программа обеспечивает простой и интуитивно понятный интерфейс для распознавания рукописных цифр с помощью сверточной нейронной сети.


Начало
БД моделей образов
Рукописные ввод данных
Предварительная обработка исходных данных
Получение и отбор признаков
Выбор модели образа X_i
Обнаружение и распознавание образа, соответствующего модели X_i
Нет
Проверена вся область данных
Нет
Да
Перебраны все модели образов
Да
Сбор всех выявленных признаков с установленной вероятностью
Сравнение и обработка результата, определение вероятности ошибки
Конец

3.4 Описание кода

Вначале импортируются необходимые библиотеки для разработки пользовательского интерфейса (PyQt5), обработки изображений (PIL, Pillow), работы с данными (numpy, torch, torchvision) и визуализации (matplotlib).

```
class Canvas(QtWidgets.QLabel):
    def __init__(self):
        super().__init__()
        pixmap = QtGui.QPixmap(560, 560)
        pixmap.fill(QtGui.QColor("white"))
        self.setPixmap(pixmap)
```

Этот класс Canvas определяет холст для рисования цифр. Он наследуется от QLabel и предоставляет функциональность для рисования с помощью событий мыши.

```
class MainWindow(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()
        try:
            self.model = torch.load('/home/and/Python/sergey/New_s/DigitRecognition/my_mnist_model777.pt')
        except:
            msg = QMessageBox()
            msg.setWindowTitle("ERROR")
            msg.setText("Please train a model before trying to recognize digits")
            x = msg.exec_()
```

Класс MainWindow представляет основное окно приложения. Он загружает предварительно обученную модель (my_mnist_model777.pt), если таковая имеется, в противном случае отображает сообщение об ошибке.

```
class ScrollMessageBox(QMessageBox):
    def __init__(self, l, title, *args, **kwargs):
        QMessageBox.__init__(self, *args, **kwargs)
```

```
            scroll = QScrollArea(self)
            scroll.setWidgetResizable(True)
            QMessageBox.setWindowTitle(self, title)
            self.content = QWidget()
            scroll.setWidget(self.content)
            lay = QVBoxLayout(self.content)
            for item in l:
                img = QLabel()
                qimage = ImageQt(item[0])
                img.setPixmap(QPixmap.fromImage(qimage))
                lay.addWidget(img)
            self.layout().addWidget(scroll, 0, 0, 1,
  self.layout().columnCount())
            self.setStyleSheet("QScrollArea{min-width:300  px;  min-
height: 400px}")
```

Класс ScrollMessageBox определяет диалоговое окно с полосой прокрутки для отображения изображений. Он наследуется от QMessageBox и используется для визуализации изображений из набора данных MNIST.

```
class StringBox(QSpinBox):
    def __init__(self):
        super().__init__()
        strings = ["Model 1", "Model 2"]
        self.setStrings(strings)
```

Класс StringBox определяет выпадающий список с текстовыми опциями. Он используется для переключения между различными моделями в пользовательском интерфейсе.

```
app = QtWidgets.QApplication(sys.argv)
window = MainWindow()
window.show()
app.exec_()
```

Наконец, код создает экземпляр QApplication, создает MainWindow, отображает его и запускает цикл обработки событий приложения. Эта часть важна для запуска приложения с графическим интерфейсом.

Обучение модели(ai_with_class.py):

```
import numpy as np
import torch
import torchvision
from time import time
from torchvision import datasets, transforms
from torch import nn, optim
```

Эти строки кода импортируют необходимые библиотеки для работы с данными, нейронными сетями и оптимизацией. Библиотека numpy используется для работы с массивами данных, torch - для работы с тензорами и нейронными сетями, а torchvision предоставляет функции для работы с наборами данных, такими как MNIST.

```
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize((0.5,), (0.5,)),
                                ])
```

Эта часть определяет преобразование (transform), которое будет применяться к загруженным изображениям. Здесь используется transforms.Compose для объединения нескольких преобразований: transforms.ToTensor() преобразует изображения в тензоры, а transforms.Normalize() нормализует значения пикселей изображений.

```
trainset = datasets.MNIST(root='mnistdata/', download=True, train=True, transform=transform)
valset = datasets.MNIST(root='mnistdata/', download=True, train=False, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64, shuffle=True)
valloader = torch.utils.data.DataLoader(valset, batch_size=64, shuffle=True)
```

Этот блок кода загружает и подготавливает данные MNIST для обучения (trainset) и валидации (valset). datasets.MNIST загружает набор данных MNIST, train=True указывает, что используются данные обучения, а train=False - для данных валидации. torch.utils.data.DataLoader создает итераторы для этих наборов данных, позволяя эффективно загружать данные в модель.

```
input_size = 784
hidden_sizes = [128, 64]
output_size = 10
```

Эти переменные определяют размеры входного слоя (input_size), скрытых слоев (hidden_sizes) и выходного слоя (output_size) нейронной сети.

```
model = nn.Sequential(nn.Linear(input_size, hidden_sizes[0]),
                      nn.ReLU(),
                      nn.Linear(hidden_sizes[0],
hidden_sizes[1]),
                      nn.ReLU(),
                      nn.Linear(hidden_sizes[1], output_size),
                      nn.LogSoftmax(dim=1))
```

Этот блок определяет архитектуру нейронной сети. Используется nn.Sequential, чтобы последовательно объединить слои. В этой нейронной сети есть три полносвязных слоя (nn.Linear) и два слоя активации ReLU (nn.ReLU). Функция активации nn.LogSoftmax применяется к последнему слою для предсказания вероятностей принадлежности к классам.

```
criterion = nn.NLLLoss()
optimizer = optim.SGD(model.parameters(),lr=0.003, momentum=0.9)
```

Здесь определяются функция потерь (criterion) и оптимизатор (optimizer). Функция потерь nn.NLLLoss() используется для вычисления ошибки предсказания, а оптимизатор optim.SGD используется для обновления весов модели методом стохастического градиентного спуска (SGD) с моментом.

```
time0 = time()
epochs = 15
print("Training for up to epoch {}.".format(epochs - 1))
```

```
for e in range(epochs):
    running_loss = 0
    for images, labels in trainloader:
        images = images.view(images.shape[0], -1)
        optimizer.zero_grad()
        output = model(images)
        loss = criterion(output, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    else:
        print("Epoch {} - Training loss: {}".format(e, running_loss/len(trainloader)))
print("\nTraining Time (in minutes) =", (time()-time0)/60)
```

В этом блоке выполняется обучение нейронной сети на данных обучения. Внешний цикл по эпохам (for e in range(epochs)) выполняет обучение на протяжении указанного количества эпох. Внутренний цикл перебирает батчи из загруженных данных и обновляет веса модели на каждом шаге с помощью обратного распространения ошибки.

```
correct_count, all_count = 0, 0
for images, labels in valloader:
    for i in range(len(labels)):
        img = images[i].view(1, 784)
        with torch.no_grad():
            logps = model(img)
        ps = torch.exp(logps)
        probab = list(ps.numpy()[0])
        pred_label = probab.index(max(probab))
        true_label = labels.numpy()[i]
        if true_label == pred_label:
            correct_count += 1
        all_count += 1
```

Результатом этого цикла является оценка точности модели на данных валидации. Для каждого изображения из набора данных валидации модель предсказывает метку класса и сравнивает ее с истинной меткой. Точность вычисляется как количество правильно предсказанных изображений, деленное на общее количество изображений.

```
def getAccuracy():
    return correct_count / all_count

print("Number Of Images Tested =", all_count)
print("\nModel Accuracy =", (correct_count / all_count))

def getAccuracy():
    return correct_count / all_count

print("Number Of Images Tested =", all_count)
print("\nModel Accuracy =", (correct_count / all_count))
```

Этот блок кода определяет функцию getAccuracy(), которая возвращает точность модели. Затем вычисленные значения точности выводятся на экран.

```
torch.save(model,
'/home/and/Python/sergey/New_s/DigitRecognition/my_mnist_model777.pt
')
print("\nModel saved")
```

Последний блок кода сохраняет обученную модель в файл. Функция torch.save() принимает два аргумента: модель, которую необходимо сохранить, и путь к файлу, в котором будет сохранена модель. В данном случае модель сохраняется в файле под именем my_mnist_model777.pt в указанной директории.

После успешного сохранения модели выводится сообщение "Model saved".

4 Результаты тестирования

4.1 Успешные результаты распознования

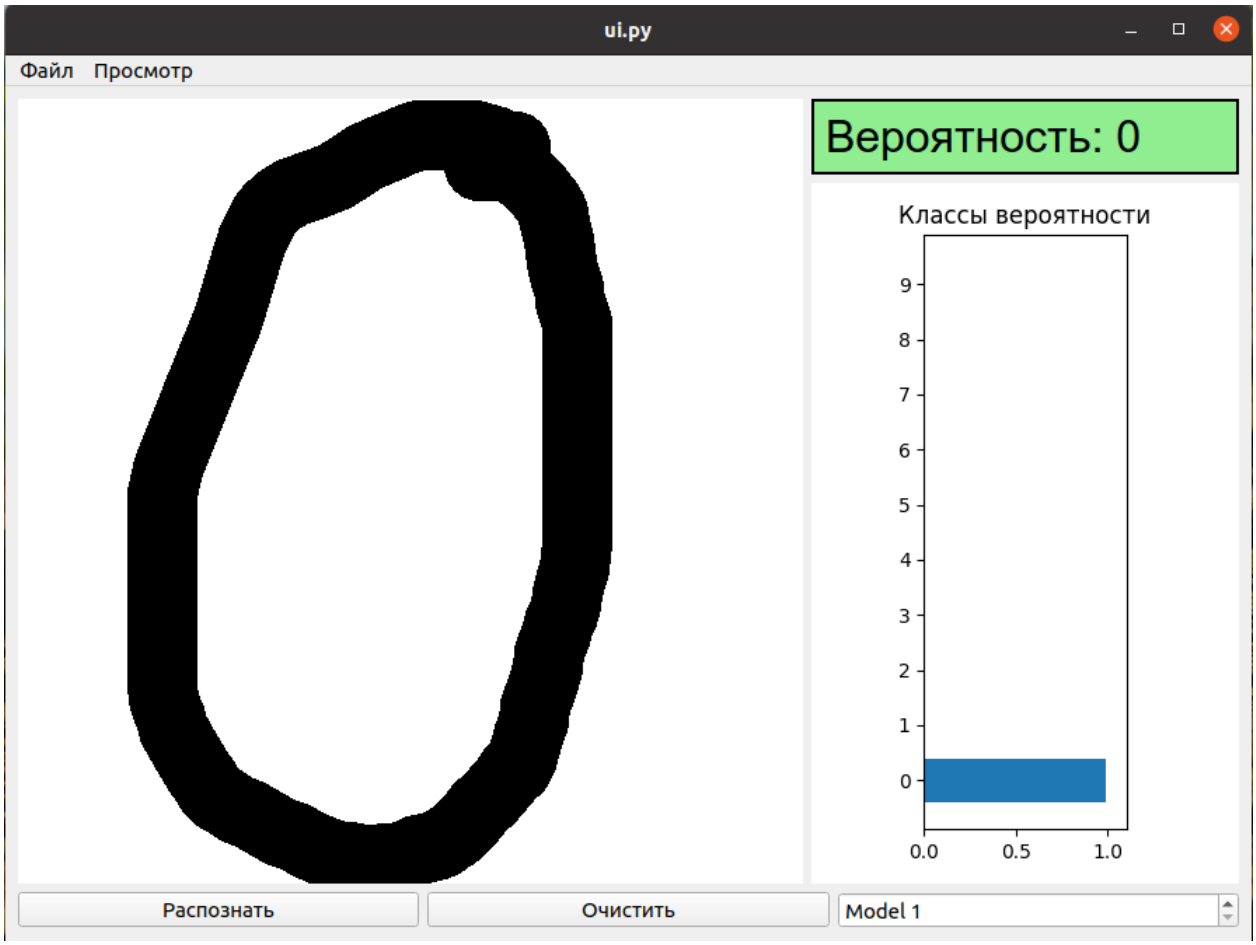


Рисунок 18 – Тест 1

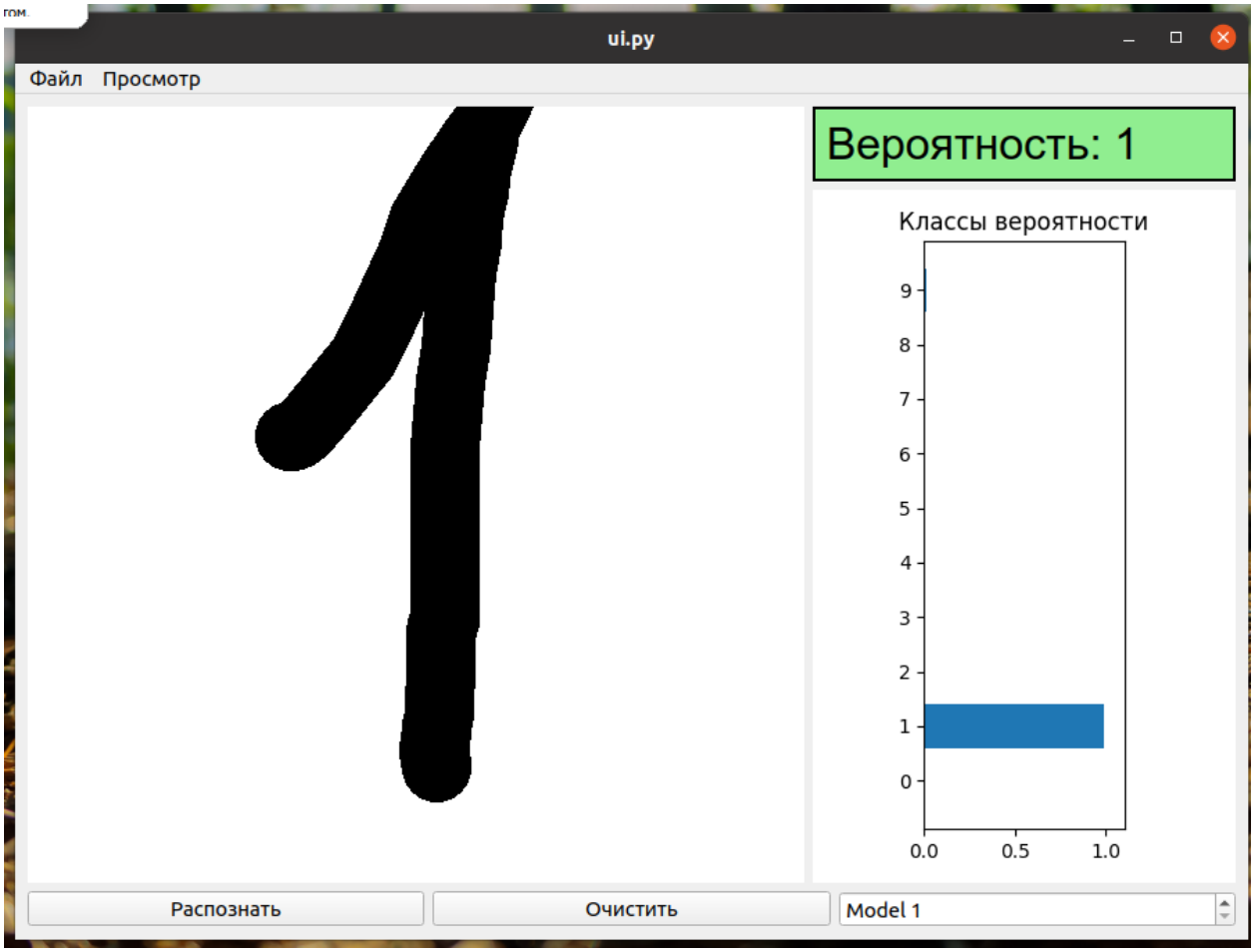


Рисунок 19 – Тест 2

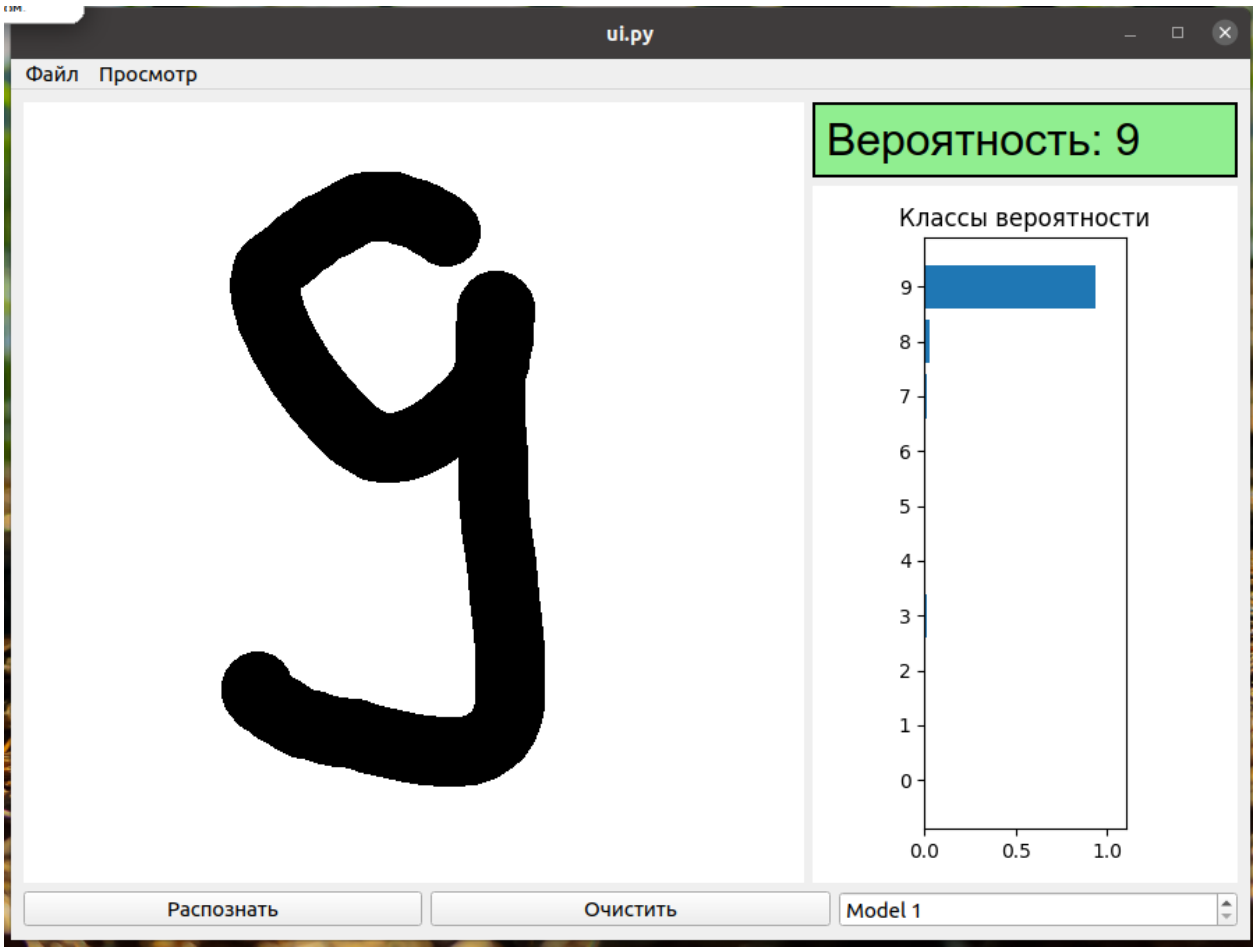


Рисунок 20 – Тест 3

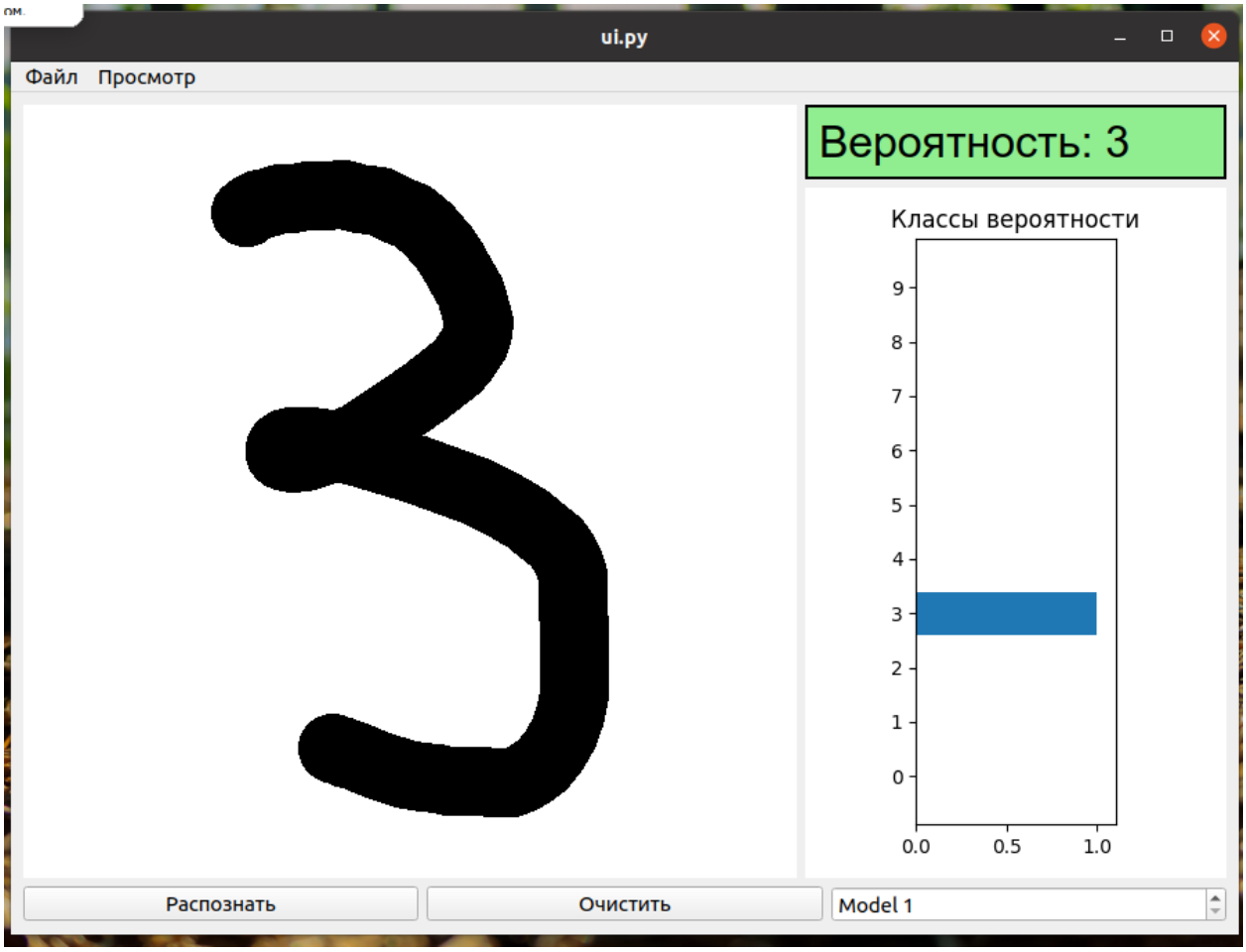


Рисунок 21 – Тест 4

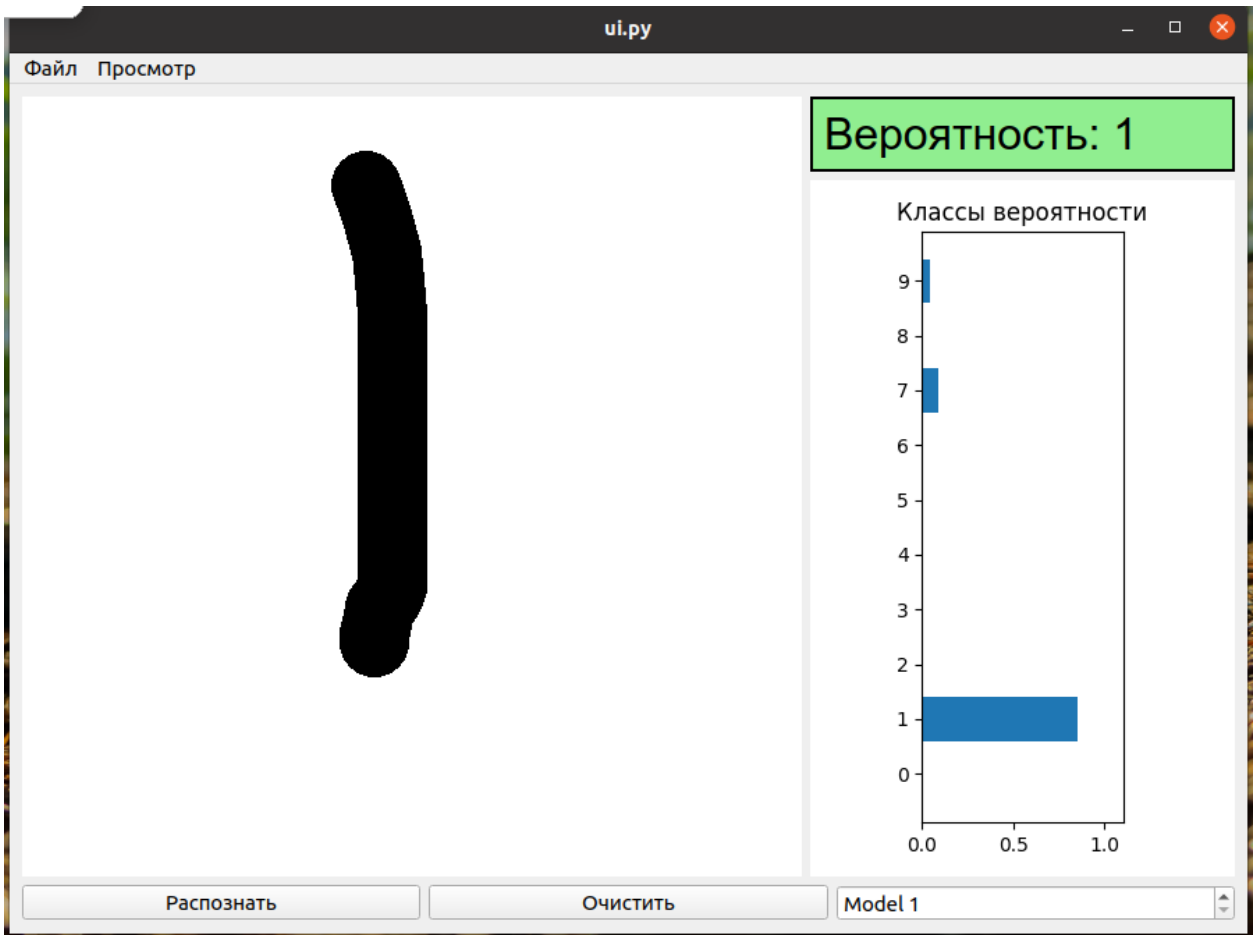


Рисунок 22 - Тест 5

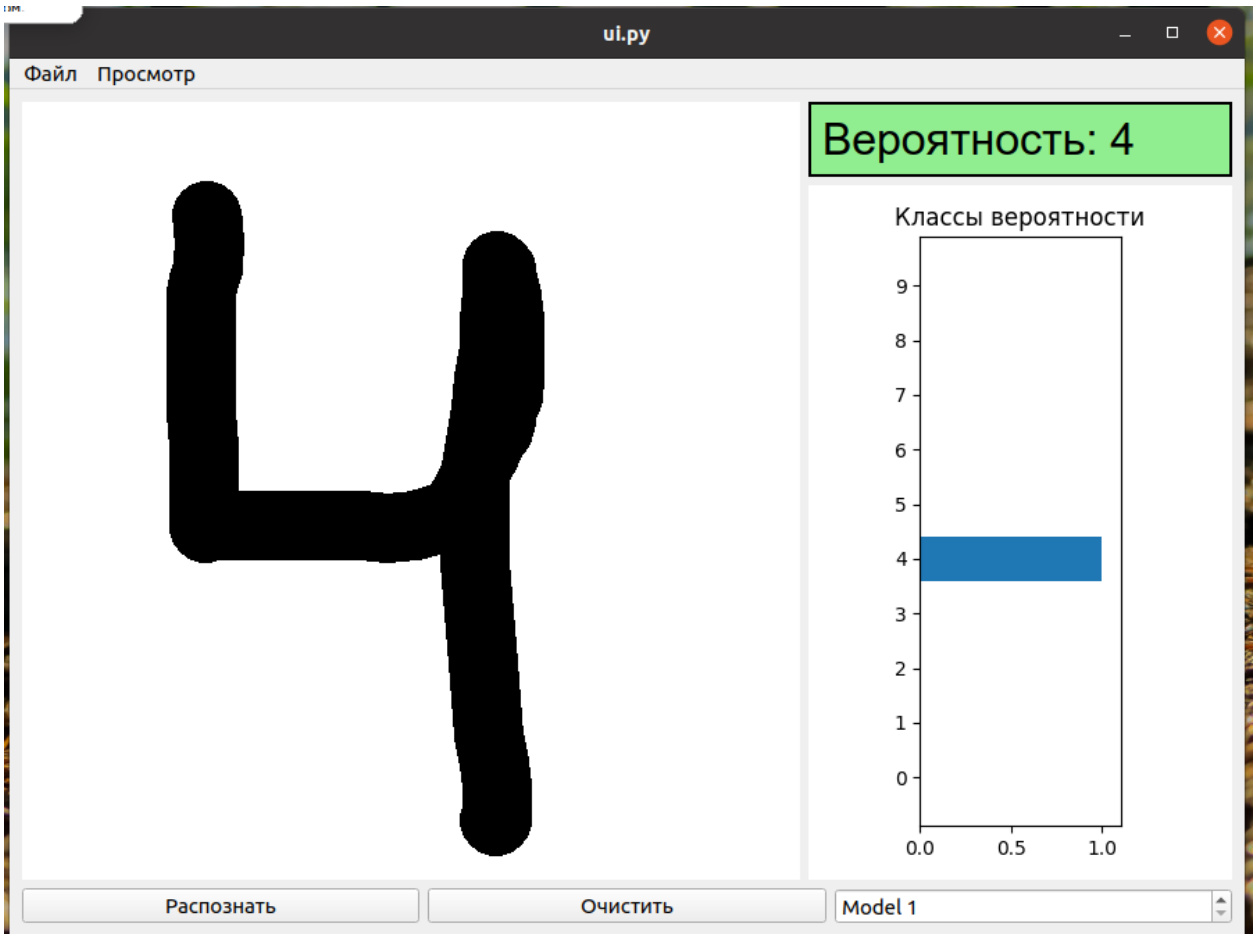


Рисунок 23 - Тест 6

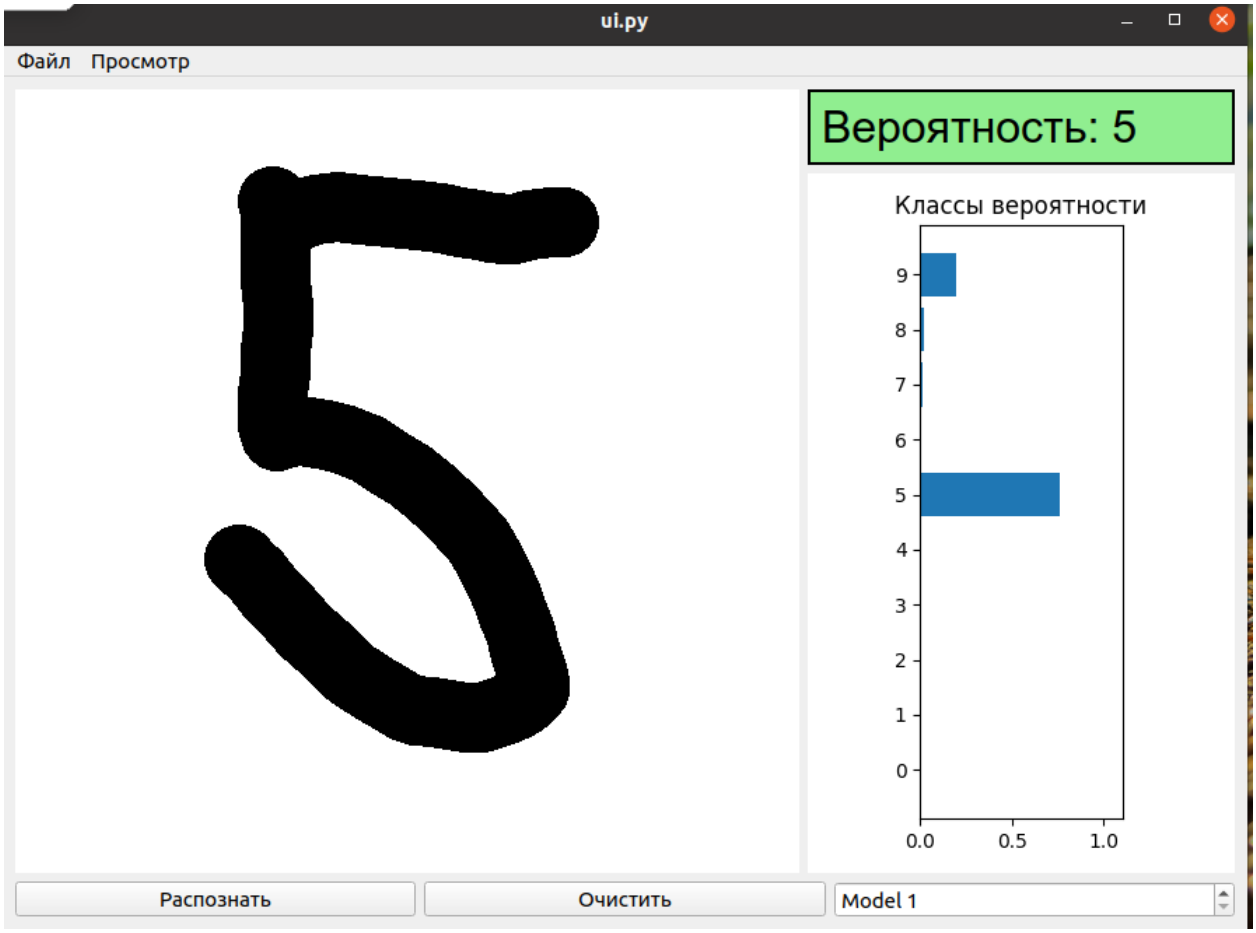


Рисунок 24 - Тест 7

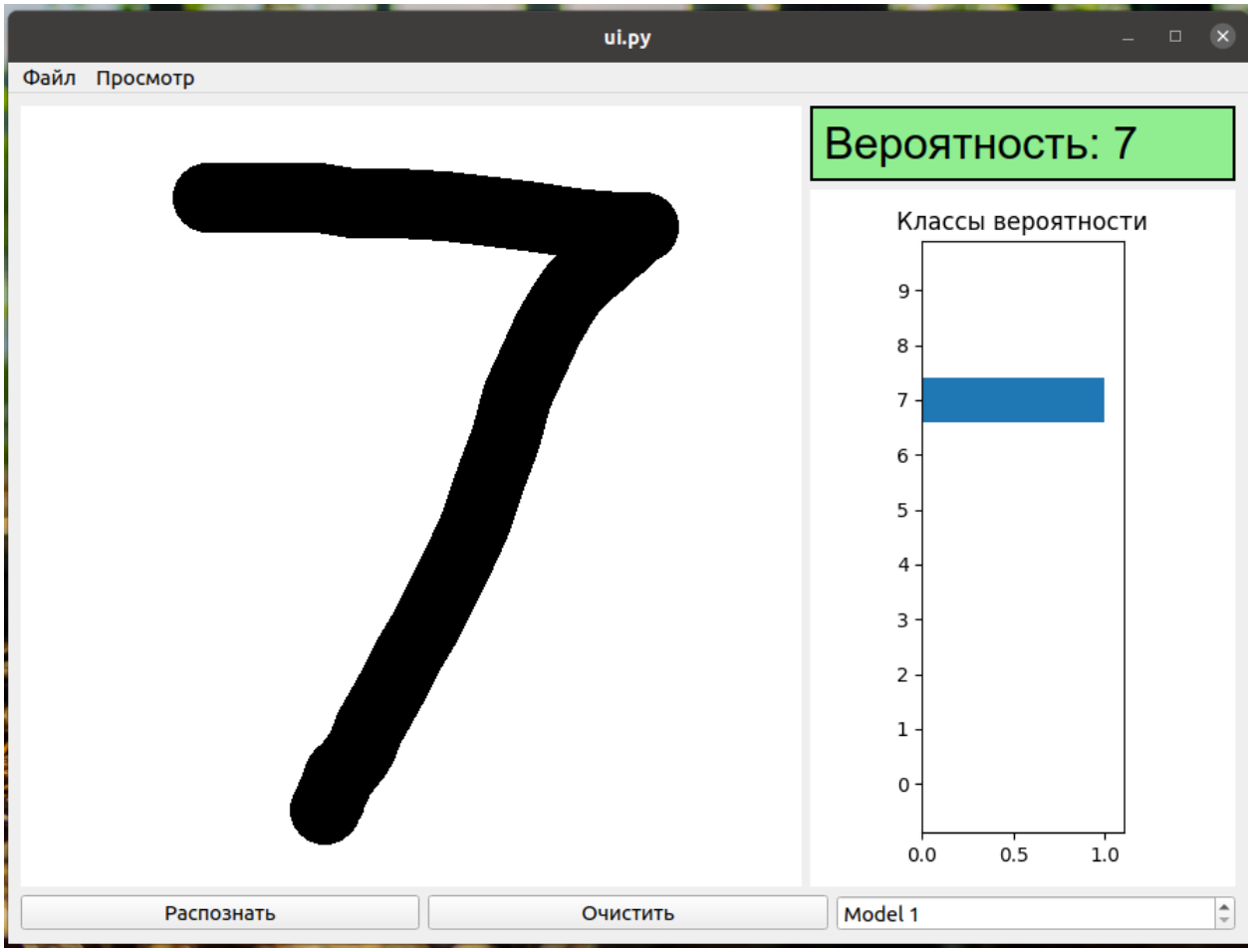


Рисунок 25 – Тест 8

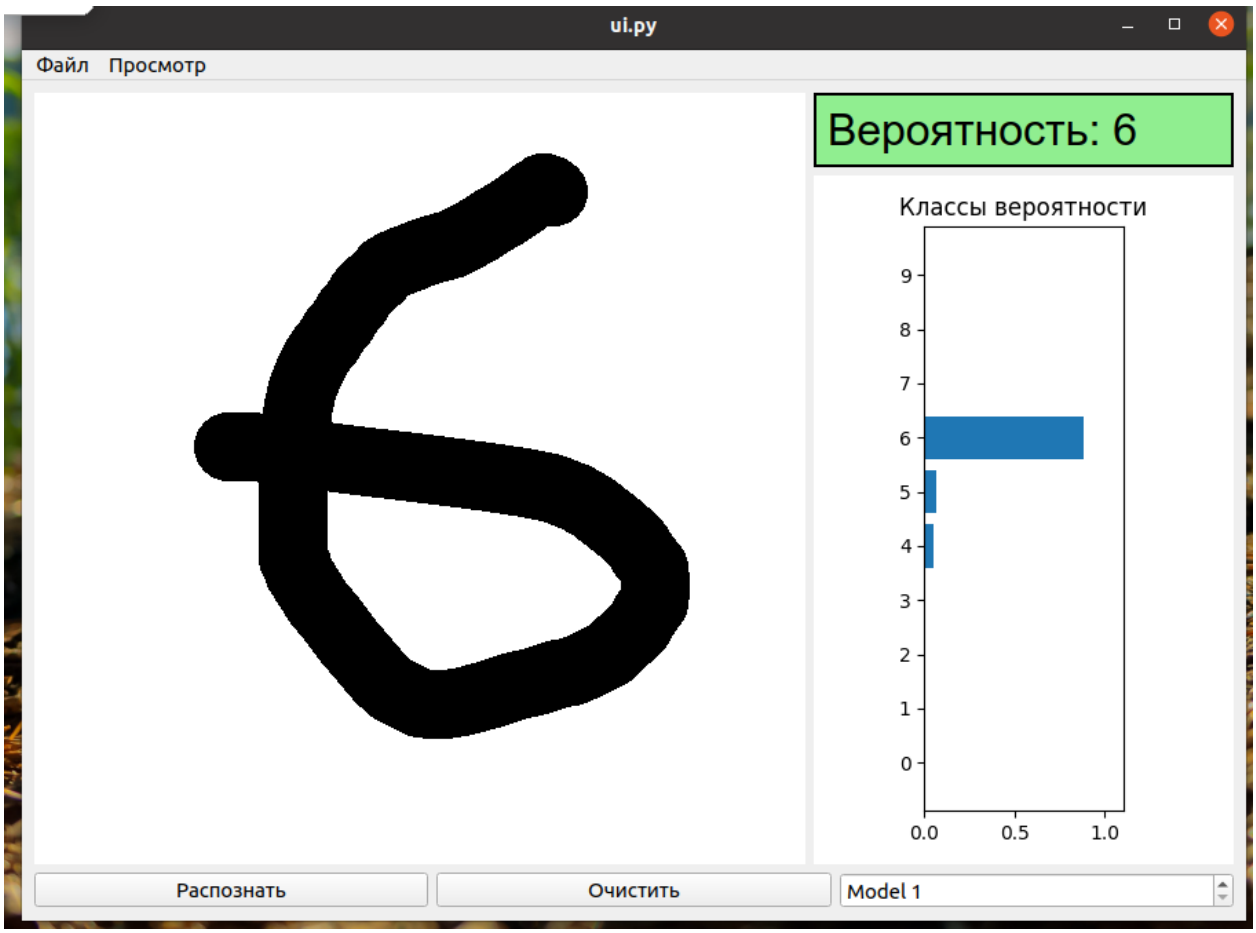


Рисунок 26 – Тест 9 2

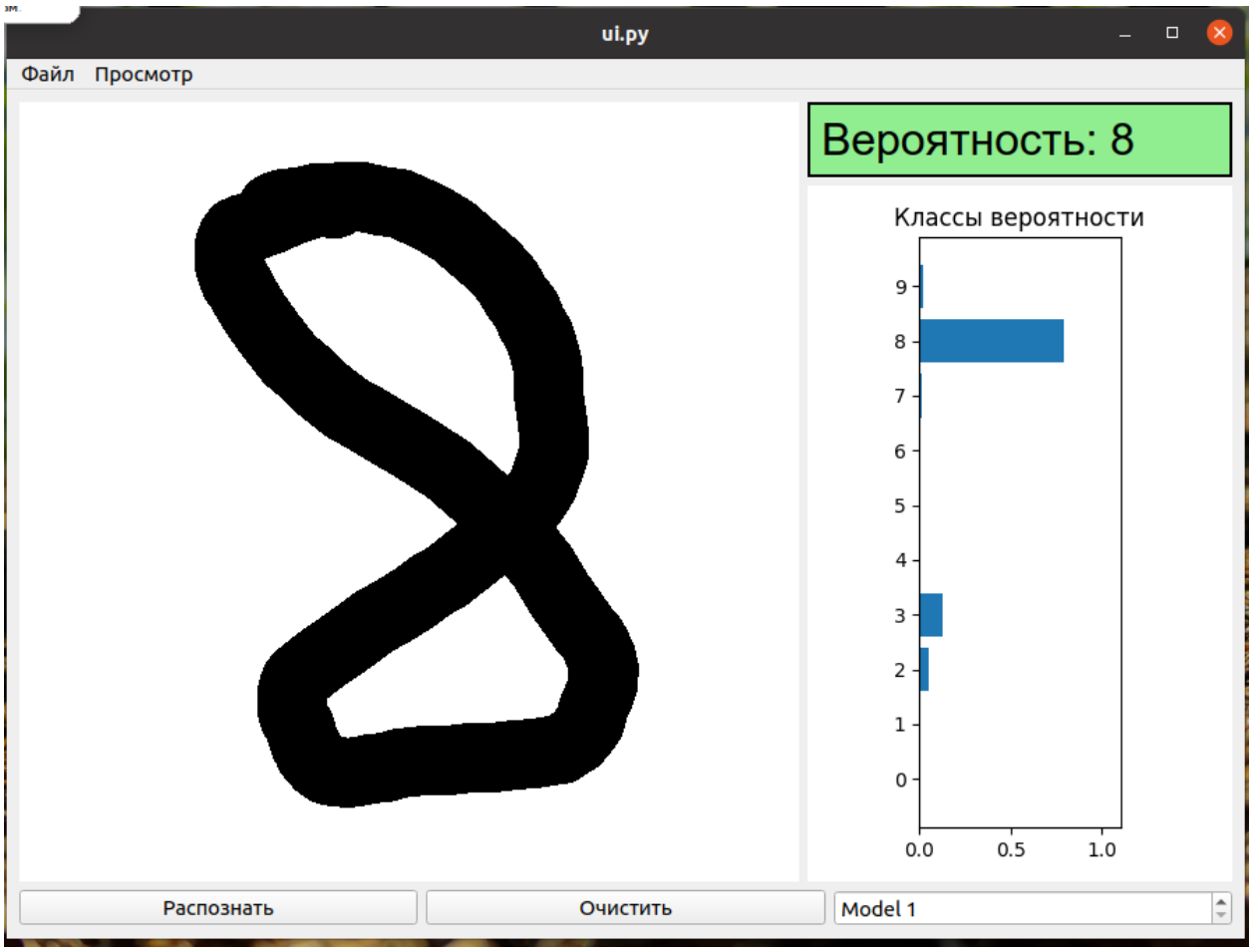


Рисунок 27 – Тест 10

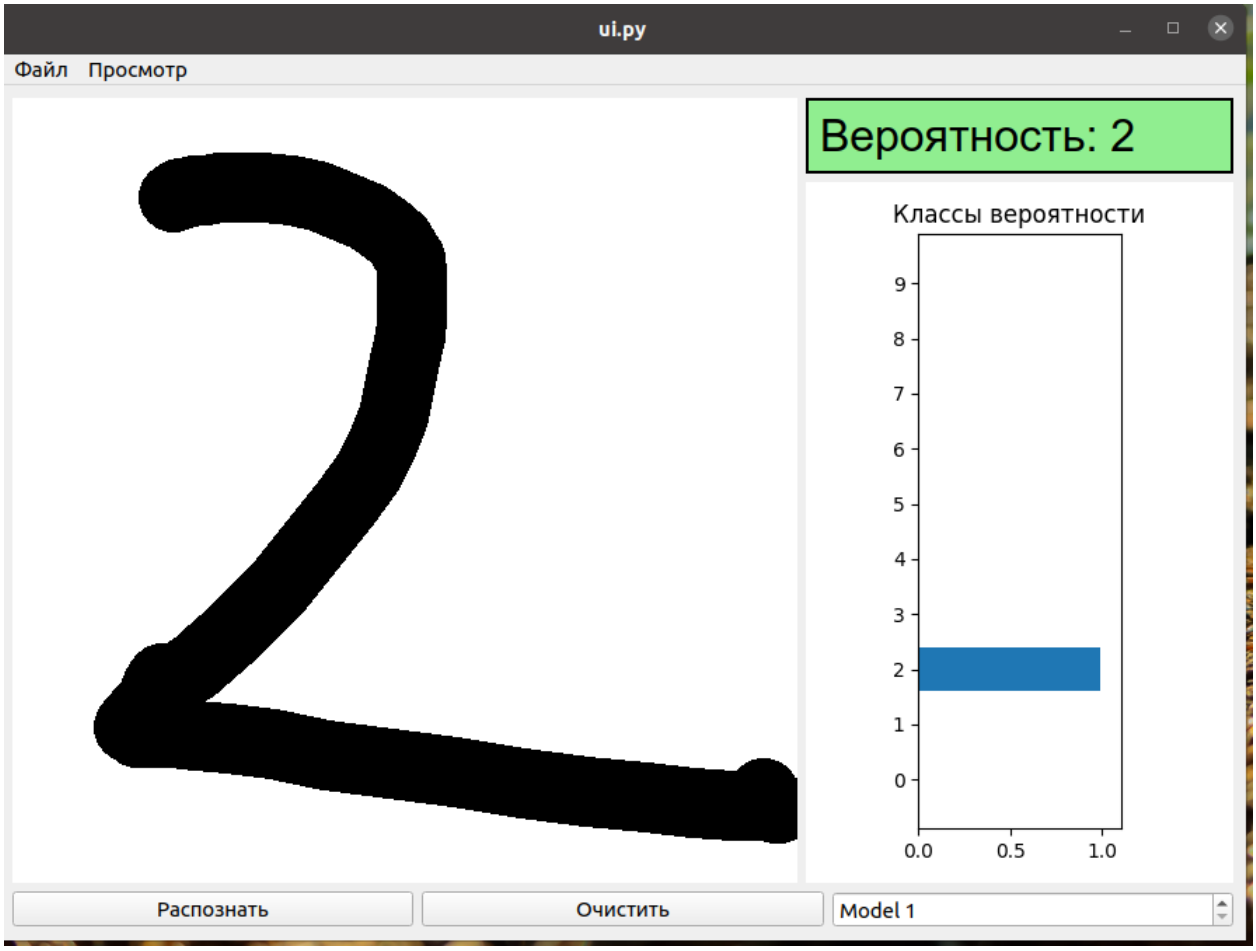


Рисунок 28 – Тест 11

4.2 Ошибочные результаты распознования

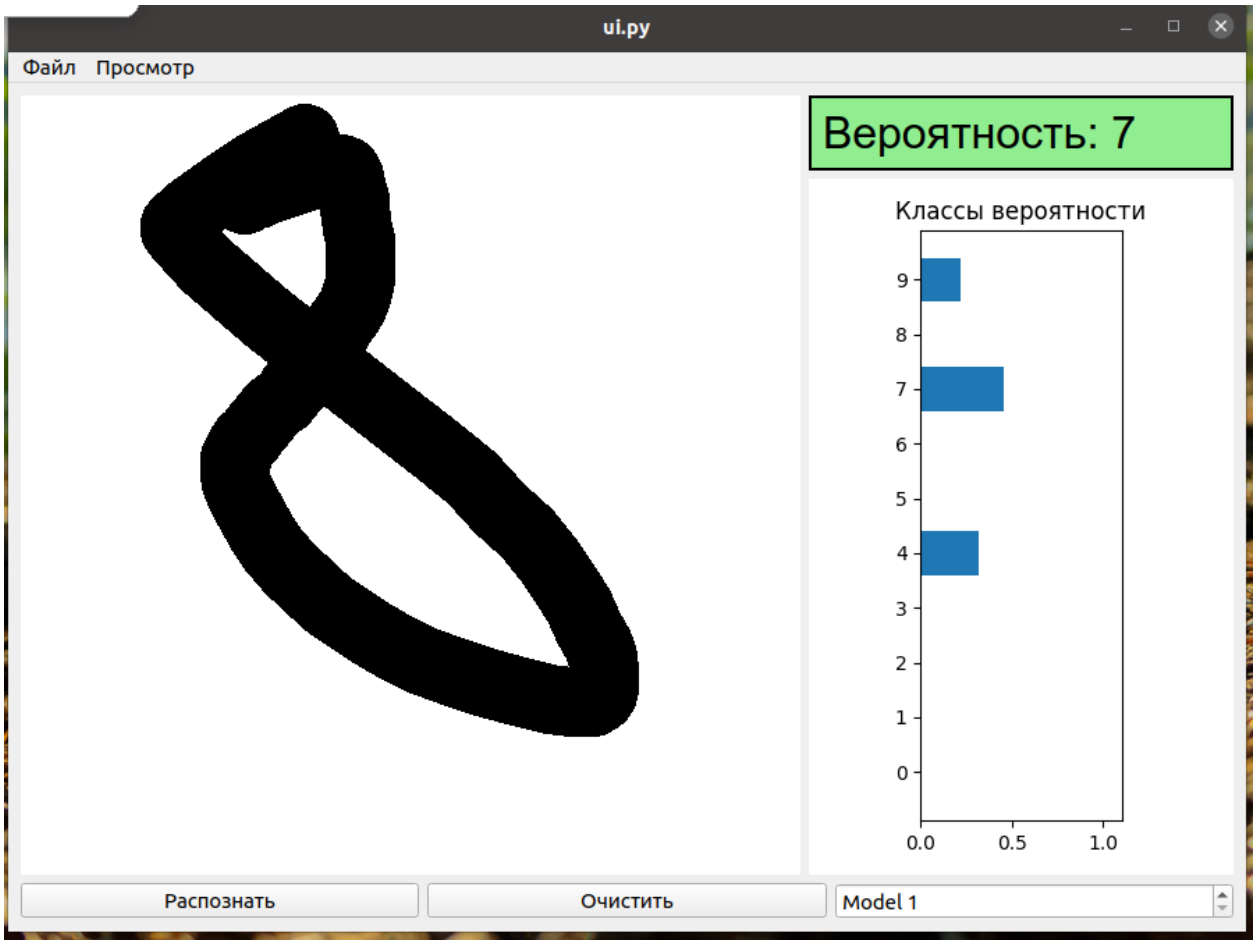


Рисунок 29 - Тест 12

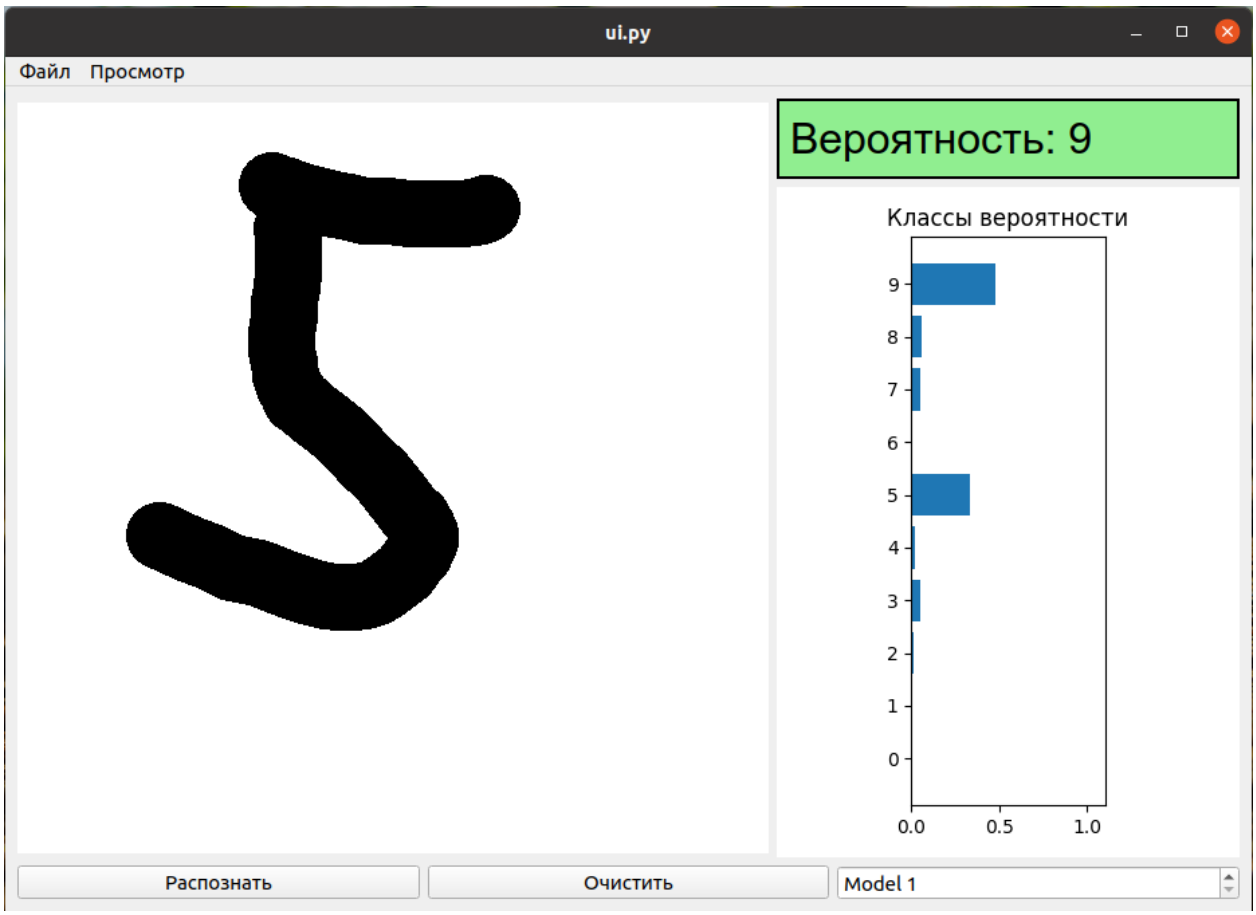


Рисунок 30 - Тест 13

Как видно из результатов тестирования, модели сверточных нейронных сетей и трансформера имеют высокую точность распознования и небольшую вероятность ошибки. Так же удалось добиться практически моментального распознования цифры.

Трансфомер – это относительно новая технология по сравнению со сверточной нейронной сетью и показала отличные результаты в распозновании. Эти две архитектуры были выбраны исходя из того, что они обе имеют большую точность в распозновании обьектов на изображении и трансформер в ряде случаев работает точнее чем CNN. Такая точность выше по причиче применения трансформером механизма внимания, в то время как у CNN свертки затрудняют моделирование долгосрочных зависимостей.

Тем неменее результаты полностью удоволетворяют поставленной задаче и пользователь может выбрать какую модель использовать и быстро и точно получить результат.

ЗАКЛЮЧЕНИЕ

В данной работе было разработано приложение для распознавания цифр с использованием сверточной нейронной сети и трансформера. Программа предоставляет удобный и интуитивно понятный интерфейс, позволяющий пользователям нарисовать цифру на планшете и автоматически определить ее.

Одной из ключевых особенностей разработанного приложения является использование библиотек PyQt5 для создания графического интерфейса и Keras для работы с нейронными сетями. Это обеспечивает высокую производительность и эффективность в распознавании цифр.

Программа применяет алгоритм выполнения скриншота для захвата изображения цифры, после чего преобразует его в формат, подходящий для обработки нейронной сетью. Нейронная сеть, обученная на размеченных данных, проводит классификацию цифры и выводит результат на экран.

Таким образом, разработанное приложение представляет собой эффективный инструмент для распознавания рукописных цифр с помощью технологии искусственного интеллекта. Его простота использования и точность делают его полезным инструментом в различных сферах, включая образование, банковское дело, медицину и другие. В будущем возможно расширение функциональности программы и ее применение в новых областях, что сделает ее еще более полезной и востребованной.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Qurashi A. W., Holmes V., Johnson A. P. Document processing: Methods for semantic text similarity analysis //2020 International Conference on INnovations in Intelligent SysTems and Applications (INISTA). – IEEE, 2020. – С. 1-6.
2. Pradhan N. et al. A Review on Text Similarity Technique used in IR and its Application //International Journal of Computer Applications. – 2015. – Т. 120. – №. 9. – С. 29-34.
3. Российская государственная библиотека [Электронный ресурс]. URL: https://www.rsl.ru/ (дата обращения: 20.04.2024)
4. UMAP [Электронный ресурс]. URL: https://umap-learn.readthedocs.io/en/latest/index.html (дата обращения: 21.04.2024)
5. Бенджио Иошуа, Гудфеллоу Ян. Глубокое обучение. — ДМК Пресс, 2018. — 652 с.
6. Yaser S. Abu-Mostafa, Malik Magdon-Ismail, Hsuan-Tien Lin. Learning From Data. — AMLBook, 2012. — 213 с.
7. PyTorch [Электронный ресурс]. URL: https://pytorch.org/ (дата обращения: 05.11.2023)
8. Подготовка данных для машинного обучения [Электронный ресурс]. URL:https://machinelearningmastery.ru/data-preparation-for-machine-learning-cleansing-transformation-feature-engineering-d2334079b06d/(дата обращения: 05.06.2023)
9. Шолле Франсуа, Глубокое обучение на Python 2-е межд. издание. – СПб.: Питер, 2023 — 576 с .
10. Daniel Graupe, Deep Learning Neural Networks. Design and Case Studies. — World Scientific Publishing Company, 2016. — 280 с.

ПРИЛОЖЕНИЕ А

(обязательное)

Использование нейронных сетей для распознавания объектов на изображениях

Листинг программы

ПРИЛОЖЕНИЕ А.1

Файл «ai_with_class.py»

```
import numpy as np
import torch
import torchvision
from time import time
from torchvision import datasets, transforms
from torch import nn, optim

# defines transform function for the dataset used when
importing/downloading the dataset below
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize((0.5,),
(0.5,)),
                                ])

# downloads the datasets
trainset = datasets.MNIST(root='mnistdata/', download=True,
train=True, transform=transform)
valset = datasets.MNIST(root='mnistdata/', download=True,
train=False, transform=transform)
# loads the datasets
trainloader = torch.utils.data.DataLoader(trainset,
batch_size=64, shuffle=True)
valloader = torch.utils.data.DataLoader(valset, batch_size=64,
shuffle=True)

input_size = 784
hidden_sizes = [128, 64]
output_size = 10

# The training model
model = nn.Sequential(nn.Linear(input_size, hidden_sizes[0]),
                      nn.ReLU(),
                      nn.Linear(hidden_sizes[0],
hidden_sizes[1]),
                      nn.ReLU(),
                      nn.Linear(hidden_sizes[1], output_size),
                      nn.LogSoftmax(dim=1))

criterion = nn.NLLLoss()

optimizer = optim.SGD(model.parameters(), lr=0.003,
momentum=0.9)
time0 = time()
epochs = 15
# training for number of epochs below
print("Training for up to epoch {}.".format(epochs - 1))
for e in range(epochs):
```

```
    running_loss = 0
    for images, labels in trainloader:
        # Flatten MNIST images into a 784 long vector
        images = images.view(images.shape[0], -1)

        # Training pass
        optimizer.zero_grad()

        output = model(images)
        loss = criterion(output, labels)

        # This is where the model learns by backpropagating
        loss.backward()

        # And optimizes its weights here
        optimizer.step()

        running_loss += loss.item()
    else:
        print("Epoch {} - Training loss: {}".format(e,
running_loss/len(trainloader)))
print("\nTraining Time (in minutes) =", (time()-time0)/60)

# Calculating the accuracy of our model
correct_count, all_count = 0, 0
for images, labels in valloader:
    for i in range(len(labels)):
        img = images[i].view(1, 784)
        with torch.no_grad():
            logps = model(img)

        ps = torch.exp(logps)
        probab = list(ps.numpy()[0])
        pred_label = probab.index(max(probab))
        true_label = labels.numpy()[i]
        if true_label == pred_label:
            correct_count += 1
        all_count += 1

# returns the accuracy of our model
def getAccuracy():
    return correct_count / all_count

print("Number Of Images Tested =", all_count)
print("\nModel Accuracy =", (correct_count / all_count))

torch.save(model,
'/home/and/Python/sergey/New_s/DigitRecognition/my_mnist_model77
7.pt')
print("\nModel saved")
```

ПРИЛОЖЕНИЕ А.2

Файл «ui_ru.py»

```
import sys
from PyQt5 import QtCore, QtGui, QtWidgets, uic
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import *
from PyQt5.QtGui import *
from PyQt5.QtCore import *
import torch
from torchvision import datasets, transforms
from PIL import Image
import PIL.ImageOps
from PIL import ImageQt
import io
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
matplotlib.use('Qt5Agg')
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg
as FigureCanvas
from matplotlib.figure import Figure
import random

# this is a custom class which handles the drawable canvas. code
is largely sourced from https://www.mfitzp.com/tutorials/bitmap-
graphics/?fbclid=IwAR0YMH_wldpyl8NFJJ7pkKXuEpEHhkeypR6ileJa22pyE
0DuIhHyOCopCDI with a few minor tweaks.
class Canvas(QtWidgets.QLabel):

    def __init__(self):
        super().__init__()
        # setup the canvas pixmap
        pixmap = QtGui.QPixmap(560, 560)
        # fill the canvas with white colour
        pixmap.fill(QtGui.QColor("white"))
        self.setPixmap(pixmap)

        self.last_x, self.last_y = None, None
        self.pen_color = QtGui.QColor('#000000')

    def mouseMoveEvent(self, e):
        if self.last_x is None: # First event.
            self.last_x = e.x()
            self.last_y = e.y()
            return # Ignore the first time.

        painter = QtGui.QPainter(self.pixmap())
        p = painter.pen()
```

```
        p.setWidth(50)
        p.setColor(self.pen_color)
        p.setCapStyle(QtCore.Qt.RoundCap)
        p.setJoinStyle(Qt.RoundJoin)
        painter.setPen(p)
        painter.drawLine(self.last_x, self.last_y, e.x(), e.y())
        painter.end()
        self.update()

        # Update the origin for next time.
        self.last_x = e.x()
        self.last_y = e.y()

    def mouseReleaseEvent(self, e):
        self.last_x = None
        self.last_y = None

class MainWindow(QtWidgets.QMainWindow):

    def __init__(self):
        super().__init__()

        # This initializes the model being used. If non
existent, throws an error message.
        try:
            self.model =
torch.load('/home/and/Python/sergey/New_s/DigitRecognition/my_mn
ist_model777.pt')
        except:
            msg = QMessageBox()
            msg.setWindowTitle("ERROR")
            msg.setText("Please train a model before trying to
recognize digits")
            x = msg.exec_()

        # Code for graph Below:
        # a figure instance to plot on

        self.figure = plt.figure(figsize=(2,5))
        self.prediction = " "

        #this is the Canvas Widget that displays the 'figure'
different from the drawable canvas
        # it takes the 'figure' instance as a parameter to
__init__
        self.canvasPlot = FigureCanvas(self.figure)

        # creates the figure plot/axis
        ax = self.figure.add_subplot(111)
        ax.barh((0, 1, 2, 3, 4, 5, 6, 7, 8,
9),(0,0,0,0,0,0,0,0,0,0))
```

```
        ax.set_aspect(0.3)
        ax.set_yticks(np.arange(10))
        ax.set_yticklabels(np.arange(10))
        ax.set_title('Классы вероятности')
        ax.set_xlim(0, 1.1)
        plt.tight_layout()

        self.figure.add_subplot(ax)

        # refresh canvas
        self.canvasPlot.draw()

        # Code for other features below:
        # Setup the DRAWING Canvas
        self.canvas = Canvas()

        # Setup the layout of the application below
        w = QtWidgets.QWidget()
        l = QtWidgets.QVBoxLayout()
        w.setLayout(l)
        hor2 = QtWidgets.QHBoxLayout()
        l.addLayout(hor2)
        hor2.addWidget(self.canvas)

        vert = QtWidgets.QVBoxLayout()
        hor2.addLayout(vert)
        label = "Вероятность: "

        # creates the prediction label in the top right of gui
for displaying the predicted result
        self.classProbabilityLabel = QLabel(label)
        self.classProbabilityLabel.setStyleSheet("background-
color: lightgreen; border: 2px solid black;")
        self.classProbabilityLabel.setFont(QFont('Arial',24))

        vert.addWidget(self.classProbabilityLabel)
        vert.addWidget(self.canvasPlot)

        buttonsLayout = QtWidgets.QHBoxLayout()
        l.addLayout(buttonsLayout)

        # creates the 3 buttons and adds them to the bottom of
the gui
        clearBut = QPushButton("Очистить")
        self.modelBut = StringBox()
        recognizeBut = QPushButton("Распознать")
        buttonsLayout.addWidget(recognizeBut)
        buttonsLayout.addWidget(clearBut)
        buttonsLayout.addWidget(self.modelBut)

        self.setCentralWidget(w)
```

```
        # creating menu bar
        mainMenu = self.menuBar()

        # creating "file" and "view" menus
        fileMenu = mainMenu.addMenu("Файл")
        viewMenu = mainMenu.addMenu("Просмотр")

        # creating clear action
        clearAction = QAction("Очистить", self)
        # adding short cut to the clear action
        clearAction.setShortcut("Ctrl+C")
        # adding clear to the file menu
        fileMenu.addAction(clearAction)
        # connecting clear to file->clear and keyboard shortcut
        clearAction.triggered.connect(self.clear)
        # connecting clear to clear button
        clearBut.clicked.connect(self.clear)

        # creating recognize action
        recognizeAction = QAction("Распознать", self)
        # adding shortcut to the recognize action
        recognizeAction.setShortcut("Ctrl+R")
        # adding recognize to the file menu
        fileMenu.addAction(recognizeAction)
        # connecting regognzie to file->recognize and keyboard
shortcut
        recognizeAction.triggered.connect(self.recognize)
        # connecting recognize to recognize button
        recognizeBut.clicked.connect(self.recognize)

        # creating training action and adding the option to the
menus
        trainModelAction = QAction("Тренировочная модель", self)
        fileMenu.addAction(trainModelAction)
        trainModelAction.triggered.connect(self.trainTheModel)

        # Creating the action for viewing training and testing
data sets
        viewTrainingImages = QAction("Просмотр тренировочных
изображений", self)
        viewMenu.addAction(viewTrainingImages)
        viewTrainingImages.triggered.connect(self.trainingPopout
)
        viewTestingImages = QAction("Просмотр тестовых
изображений", self)
        viewMenu.addAction(viewTestingImages)
        viewTestingImages.triggered.connect(self.testPopout)

    def trainTheModel(self):
```

```
        # This trains the model by importing the script
ai_with_class and running it.
        msg = QMessageBox()
        msg.setStyleSheet("QLabel{min-width: 500px; min-
height:100px}")
        msg.setText("Its training just hold on")
        x = msg.exec_()
        import ai_with_class
        self.model =
torch.load('/home/and/Python/sergey/New_s/DigitRecognition/my_mn
ist_model777.pt')

    def view_classify(self, ps):
        # sourced from
https://github.com/amitrajitbose/handwritten-digit-
recognition/blob/master/handwritten_digit_recognition_CPU.ipynb
        # clearing old figure
        self.figure.clear()

        # create an axis
        ax = self.figure.add_subplot(111)
        ps = ps.data.numpy().squeeze()
        ax.barh((0, 1, 2, 3, 4, 5, 6, 7, 8, 9), ps)
        ax.set_aspect(0.3)
        ax.set_yticks(np.arange(10))
        ax.set_yticklabels(np.arange(10))
        ax.set_title('Классы вероятности')
        ax.set_xlim(0, 1.1)
        plt.tight_layout()
        self.figure.add_subplot(ax)

        # refresh canvas
        self.canvasPlot.draw()

    def recognize(self):
        # The try except statement below checks if there are any
errors with recognizing the digit and if there is a model
currently assigned.
        # If no model is assigned, or there is an error, an
error message box will appear.
        try:
            # creates a buffer to save the canvas drawn image to
            buffer = QBuffer()
            buffer.open(QBuffer.ReadWrite)

            # saves the canvas drawn image to the buffer
            self.canvas.pixmap().save(buffer, "PNG")

            # opens the image saved in the buffer and converts
it to grayscale
```

```
            pil_im =
Image.open(io.BytesIO(buffer.data())).convert('L')

            # resizes the image to 20x20 pixels
            pil_im = pil_im.resize((20, 20))

            # creates a new white background thats 28x28 pixels
grayscale and pastes the 20x20 canvas drawn image on top
            backgroundIm = Image.new('L', (28, 28), color=
'white')
            backgroundIm.paste(pil_im, (4, 4))

            # inverts the colours of the 28x28 image to match
the MNIST dataset
            backgroundIm = PIL.ImageOps.invert(backgroundIm)

            # converts the image to a tensor and normalizes it
            img = transforms.Compose([transforms.ToTensor(),
                                      transforms.Normalize((0.5,),
(0.5,)),
                                      ])(backgroundIm)

            # this flattens the image to be ready for model
prediction
            img = img.view(1, 784)

            #predicting the class
            if self.modelBut.value() == 0:
                with torch.no_grad():
                    logps = self.model(img)
                ps = torch.exp(logps)
                probab = list(ps.numpy()[0])

            # displays the predicted probabilities on a
horizontal bar graph
            self.view_classify(ps)

            # updates teh prediction label for the digits
            self.prediction = probab.index(max(probab))
            label = "Вероятность: " + str(self.prediction)
            self.classProbabilityLabel.setText(label)
            self.update()
        except:
            message = QMessageBox()
            message.setWindowTitle("ERROR")
            message.setText("No model trained, please train a
model before trying to recognize digits")
            message.exec_()

    def clear(self):
        # fills the drawing canvas with white to clear it
```

```
        self.canvas.pixmap().fill(QtGui.QColor("white"))
        self.canvas.update()

        # clearing old figure
        self.figure.clear()
         # create an axis
        ax = self.figure.add_subplot(111)
        ax.barh((0, 1, 2, 3, 4, 5, 6, 7, 8,
9),(0,0,0,0,0,0,0,0,0,0))
        ax.set_aspect(0.3)
        ax.set_yticks(np.arange(10))
        ax.set_yticklabels(np.arange(10))
        ax.set_title('Классы вероятности:')
        ax.set_xlim(0, 1.1)
        plt.tight_layout()
        self.figure.add_subplot(ax)

        # refresh canvas
        self.canvasPlot.draw()
        self.classProbabilityLabel.setText("Вероятность:  ")

        # this downloads and shows the 60000 MNIST data training
images
    def trainingPopout(self):
        trainset = datasets.MNIST(root='../mnistdata/',
download=True, train=True)
        result = ScrollMessageBox(trainset, "Training MNIST
Dataset" , None)
        result.exec_()

        # this downloads and shows the 10000 MNIST data testing
images
    def testPopout(self):
        valset = datasets.MNIST(root='../mnistdata/',
download=True, train=False)
        result = ScrollMessageBox(valset, "Testing MNIST
Dataset" , None)
        result.exec_()

    # This is for future functionality that we didnt get around
to doing. It is currently unused.
    def downloadData(self):
        transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize((0.5,),
(0.5,)),
                                ])
        trainset = datasets.MNIST(root='../mnistdata/',
download=True, train=True, transform=transform)
        valset = datasets.MNIST(root='../mnistdata/',
download=True, train=False, transform=transform)
```

```
# This class is for the pop up window with a scroll bar used to
display the MNIST data images
# sourced from
https://stackoverflow.com/questions/47345776/pyqt5-how-to-add-a-
scrollbar-to-a-qmessagebox
class ScrollMessageBox(QMessageBox):
   def __init__(self, l,title, *args, **kwargs):
      QMessageBox.__init__(self, *args, **kwargs)
      scroll = QScrollArea(self)
      scroll.setWidgetResizable(True)
      QMessageBox.setWindowTitle(self, title)
      self.content = QWidget()
      scroll.setWidget(self.content)
      lay = QVBoxLayout(self.content)
      for item in l:
        img = QLabel()
        qimage = ImageQt(item[0])
        img.setPixmap(QPixmap.fromImage(qimage))
        lay.addWidget(img)
      self.layout().addWidget(scroll, 0, 0, 1,
self.layout().columnCount())
      self.setStyleSheet("QScrollArea{min-width:300 px; min-
height: 400px}")

# this class is used for the functionality to switch between
models
class StringBox(QSpinBox):
    def __init__(self):
        super().__init__()

        strings = ["Model 1", "Model 2"]
        self.setStrings(strings)

    def setStrings(self, strings):
        strings = list(strings)
        # making tuple from the string list
        self._strings = tuple(strings)
        # creating a dictionary
        self._values = dict(zip(strings, range(len(strings))))
        # setting range to it the spin box
        self.setRange(0, len(strings)-1)

    # overwriting the textFromValue method
    def textFromValue(self, value):
        # returning string from index
        # _string = tuple
        return self._strings[value]

app = QtWidgets.QApplication(sys.argv)
window = MainWindow()
window.show()
```

ПРИЛОЖЕНИЕ А.3

Файл «ui.py»

```
window = MainWindow()
window.show()
import sys  # Импорт стандартной библиотеки sys для работы с системными параметрами
from PyQt5 import QtCore, QtGui, QtWidgets, uic  # Импорт классов из библиотеки PyQt5 для создания пользовательского интерфейса
from PyQt5.QtCore import Qt  # Импорт класса Qt из PyQt5 для работы с константами
from PyQt5.QtWidgets import *  # Импорт всех классов из модуля QtWidgets из PyQt5
from PyQt5.QtGui import *  # Импорт всех классов из модуля QtGui из PyQt5
from PyQt5.QtCore import *  # Импорт всех классов из модуля QtCore из PyQt5
import torch  # Импорт библиотеки PyTorch для машинного обучения
from torchvision import datasets, transforms  # Импорт функций из torchvision для работы с наборами данных и преобразованиями
from PIL import Image  # Импорт класса Image из библиотеки PIL для работы с изображениями
import PIL.ImageOps  # Импорт модуля ImageOps из библиотеки PIL для работы с операциями над изображениями
from PIL import ImageQt  # Импорт класса ImageQt из библиотеки PIL для работы с изображениями
import io  # Импорт модуля io для работы с потоками данных
import numpy as np  # Импорт библиотеки NumPy для работы с массивами
import matplotlib.pyplot as plt  # Импорт модуля pyplot из библиотеки Matplotlib для построения графиков
import matplotlib  # Импорт библиотеки matplotlib для настройки параметров отображения графиков
matplotlib.use('Qt5Agg')  # Установка бэкенда Qt5Agg для Matplotlib для интеграции с PyQt5
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas  # Импорт класса FigureCanvas из Matplotlib для работы с холстом
from matplotlib.figure import Figure  # Импорт класса Figure из Matplotlib для создания фигур
import random  # Импорт модуля random для работы с случайными числами

class Canvas(QtWidgets.QLabel):

    def __init__(self):
        super().__init__()
        # Настройка холста
```

```
        pixmap = QtGui.QPixmap(560, 560)  # Создание pixmap
размером 560x560
        pixmap.fill(QtGui.QColor("white"))  # Заполнение холста
белым цветом
        self.setPixmap(pixmap)

        self.last_x, self.last_y = None, None
        self.pen_color = QtGui.QColor('#000000')  # Цвет пера
для рисования

    def mouseMoveEvent(self, e):
        if self.last_x is None:  # Первое событие.
            self.last_x = e.x()
            self.last_y = e.y()
            return  # Игнорировать первый раз.

        painter = QtGui.QPainter(self.pixmap())
        p = painter.pen()
        p.setWidth(50)  # Ширина пера
        p.setColor(self.pen_color)  # Установка цвета пера
        p.setCapStyle(QtCore.Qt.RoundCap)  # Стиль концов линии
        p.setJoinStyle(Qt.RoundJoin)  # Стиль соединения линий
        painter.setPen(p)
        painter.drawLine(self.last_x, self.last_y, e.x(), e.y())
        painter.end()
        self.update()

        # Обновление координат для следующего раза.
        self.last_x = e.x()
        self.last_y = e.y()

    def mouseReleaseEvent(self, e):
        self.last_x = None
        self.last_y = None

class MainWindow(QtWidgets.QMainWindow):

    def __init__(self):
        super().__init__()

        # Инициализация используемой модели. Если не существует,
выводится сообщение об ошибке.
        try:
            self.model =
torch.load('/home/and/Python/sergey/New_s/DigitRecognition/my_mn
ist_model777.pt')
        except:
            msg = QMessageBox()
            msg.setWindowTitle("ERROR")
            msg.setText("Please train a model before trying to
recognize digits")
```

```
            x = msg.exec_()

        # Ниже код для графика:
        self.figure = plt.figure(figsize=(2, 5))
        self.prediction = " "

        # Виджет холста для отображения графика, отличается от
рисуемого холста
        # Принимает экземпляр 'figure' в качестве параметра
__init__
        self.canvasPlot = FigureCanvas(self.figure)

        # Создание графика/оси
        ax = self.figure.add_subplot(111)
        ax.barh((0, 1, 2, 3, 4, 5, 6, 7, 8, 9), (0, 0, 0, 0, 0,
0, 0, 0, 0, 0))
        ax.set_aspect(0.3)
        ax.set_yticks(np.arange(10))
        ax.set_yticklabels(np.arange(10))
        ax.set_title('Классы вероятности')
        ax.set_xlim(0, 1.1)
        plt.tight_layout()

        self.figure.add_subplot(ax)

        # Обновление холста
        self.canvasPlot.draw()

        # Ниже код для других функций:
        # Настройка рисуемого холста
        self.canvas = Canvas()

        # Настройка макета приложения
        w = QtWidgets.QWidget()
        l = QtWidgets.QVBoxLayout()
        w.setLayout(l)
        hor2 = QtWidgets.QHBoxLayout()
        l.addLayout(hor2)
        hor2.addWidget(self.canvas)

        vert = QtWidgets.QVBoxLayout()
        hor2.addLayout(vert)
        label = "Вероятность: "

        # Создание метки для отображения предсказанного
результата
        self.classProbabilityLabel = QLabel(label)
        self.classProbabilityLabel.setStyleSheet("background-
color: lightgreen; border: 2px solid black;")
        self.classProbabilityLabel.setFont(QFont('Arial', 24))
```

```
        vert.addWidget(self.classProbabilityLabel)
        vert.addWidget(self.canvasPlot)

        buttonsLayout = QtWidgets.QHBoxLayout()
        l.addLayout(buttonsLayout)

        # Создание и добавление трех кнопок внизу
пользовательского интерфейса
        clearBut = QPushButton("Очистить")
        self.modelBut = StringBox()
        recognizeBut = QPushButton("Распознать")
        buttonsLayout.addWidget(recognizeBut)
        buttonsLayout.addWidget(clearBut)
        buttonsLayout.addWidget(self.modelBut)

        self.setCentralWidget(w)

        # Создание строки меню
        mainMenu = self.menuBar()

        # Создание меню "Файл" и "Просмотр"
        fileMenu = mainMenu.addMenu("Файл")
        viewMenu = mainMenu.addMenu("Просмотр")

        # Создание действия "Очистить"
        clearAction = QAction("Очистить", self)
        clearAction.setShortcut("Ctrl+C")  # Назначение горячей
клавиши для действия "Очистить"
        fileMenu.addAction(clearAction)
        clearAction.triggered.connect(self.clear)  # Подключение
функции "clear" к действию "Очистить"
        clearBut.clicked.connect(self.clear)  # Подключение
функции "clear" к кнопке "Очистить"

        # Создание действия "Распознать"
        recognizeAction = QAction("Распознать", self)
        recognizeAction.setShortcut("Ctrl+R")  # Назначение
горячей клавиши для действия "Распознать"
        fileMenu.addAction(recognizeAction)
        recognizeAction.triggered.connect(self.recognize)  #
Подключение функции "recognize" к действию "Распознать"
        recognizeBut.clicked.connect(self.recognize)  #
Подключение функции "recognize" к кнопке "Распознать"

        # Создание действия "Тренировочная модель" и добавление
его в меню "Файл"
        trainModelAction = QAction("Тренировочная модель", self)
        fileMenu.addAction(trainModelAction)
        trainModelAction.triggered.connect(self.trainTheModel)
```

```
        # Создание действий для просмотра тренировочных и
тестовых наборов данных
        viewTrainingImages = QAction("Просмотр тренировочных
изображений", self)
        viewMenu.addAction(viewTrainingImages)
        viewTrainingImages.triggered.connect(self.trainingPopout
)
        viewTestingImages = QAction("Просмотр тестовых
изображений", self)
        viewMenu.addAction(viewTestingImages)
        viewTestingImages.triggered.connect(self.testPopout)

    def trainTheModel(self):
        # Обучение модели, загрузка скрипта ai_with_class и
запуск его
        msg = QMessageBox()
        msg.setStyleSheet("QLabel{min-width: 500px; min-
height:100px}")
        msg.setText("Its training just hold on")
        x = msg.exec_()
        import ai_with_class
        self.model =
torch.load('/home/and/Python/sergey/New_s/DigitRecognition/my_mn
ist_model777.pt')

    def view_classify(self, ps):
        # Очистка старого графика
        self.figure.clear()

        # Создание оси
        ax = self.figure.add_subplot(111)
        ps = ps.data.numpy().squeeze()
        ax.barh((0, 1, 2, 3, 4, 5, 6, 7, 8, 9), ps)
        ax.set_aspect(0.3)
        ax.set_yticks(np.arange(10))
        ax.set_yticklabels(np.arange(10))
        ax.set_title('Классы вероятности')
        ax.set_xlim(0, 1.1)
        plt.tight_layout()
        self.figure.add_subplot(ax)

        # Обновление холста
        self.canvasPlot.draw()

    def recognize(self):
        try:
            # Создание буфера для сохранения изображения,
нарисованного на холсте
            buffer = QBuffer()
            buffer.open(QBuffer.ReadWrite)
```

```
            # Сохранение нарисованного изображения в буфере
            self.canvas.pixmap().save(buffer, "PNG")

            # Открытие изображения, сохраненного в буфере, и
преобразование его в оттенки серого
            pil_im =
Image.open(io.BytesIO(buffer.data())).convert('L')

            # Изменение размера изображения на 20x20 пикселей
            pil_im = pil_im.resize((20, 20))

            # Создание нового белого фона размером 28x28
пикселей в градации серого и вставка нарисованного изображения
размером 20x20 пикселей поверх него
            backgroundIm = Image.new('L', (28, 28),
color='white')
            backgroundIm.paste(pil_im, (4, 4))

            # Инвертирование цветов изображения размером 28x28,
чтобы совпадать с набором данных MNIST
            backgroundIm = PIL.ImageOps.invert(backgroundIm)

            # Преобразование изображения в тензор и его
нормализация
            img = transforms.Compose([transforms.ToTensor(),
                                      transforms.Normalize((0.5
,), (0.5,)),
                                      ])(backgroundIm)

            # Перевод изображения в плоский формат для
предсказания моделью
            img = img.view(1, 784)

            # Предсказание класса
            if self.modelBut.value() == 0:
                with torch.no_grad():
                    logps = self.model(img)
                ps = torch.exp(logps)
                probab = list(ps.numpy()[0])

            # Отображение предсказанных вероятностей на
горизонтальной столбчатой диаграмме
            self.view_classify(ps)

            # Обновление метки предсказания для цифр
            self.prediction = probab.index(max(probab))
            label = "Вероятность: " + str(self.prediction)
            # Обновление метки предсказания для цифр
            self.prediction = probab.index(max(probab))
            label = "Вероятность: " + str(self.prediction)
            self.classProbabilityLabel.setText(label)
```

```
            self.update()
        except:
            message = QMessageBox()
            message.setWindowTitle("ERROR")
            message.setText("No model trained, please train a
model before trying to recognize digits")
            message.exec_()

    def clear(self):
        # Заполнение рисуемого холста белым цветом для очистки
        self.canvas.pixmap().fill(QtGui.QColor("white"))
        self.canvas.update()

        # Очистка старого графика
        self.figure.clear()
        ax = self.figure.add_subplot(111)
        ax.barh((0, 1, 2, 3, 4, 5, 6, 7, 8, 9), (0, 0, 0, 0, 0,
0, 0, 0, 0, 0))
        ax.set_aspect(0.3)
        ax.set_yticks(np.arange(10))
        ax.set_yticklabels(np.arange(10))
        ax.set_title('Классы вероятности:')
        ax.set_xlim(0, 1.1)
        plt.tight_layout()
        self.figure.add_subplot(ax)

        # Обновление холста
        self.canvasPlot.draw()
        self.classProbabilityLabel.setText("Вероятность: ")

    def trainingPopout(self):
        # Загрузка и отображение 60000 тренировочных изображений
из набора данных MNIST
        trainset = datasets.MNIST(root='../mnistdata/',
download=True, train=True)
        result = ScrollMessageBox(trainset, "Training MNIST
Dataset", None)
        result.exec_()

    def testPopout(self):
        # Загрузка и отображение 10000 тестовых изображений из
набора данных MNIST
        valset = datasets.MNIST(root='../mnistdata/',
download=True, train=False)
        result = ScrollMessageBox(valset, "Testing MNIST
Dataset", None)
        result.exec_()

    # Функция для загрузки данных, которая в настоящее время не
используется
    def downloadData(self):
```

```
        transform = transforms.Compose([transforms.ToTensor(),
                                        transforms.Normalize((0
.5,), (0.5,)),
                                        ])
        trainset = datasets.MNIST(root='../mnistdata/',
download=True, train=True, transform=transform)
        valset = datasets.MNIST(root='../mnistdata/',
download=True, train=False, transform=transform)

# Класс для всплывающего окна с полосой прокрутки для
отображения изображений данных MNIST
# Источник: https://stackoverflow.com/questions/47345776/pyqt5-
how-to-add-a-scrollbar-to-a-qmessagebox
class ScrollMessageBox(QMessageBox):
    def __init__(self, l, title, *args, **kwargs):
        QMessageBox.__init__(self, *args, **kwargs)
        scroll = QScrollArea(self)
        scroll.setWidgetResizable(True)
        QMessageBox.setWindowTitle(self, title)
        self.content = QWidget()
        scroll.setWidget(self.content)
        lay = QVBoxLayout(self.content)
        for item in l:
            img = QLabel()
            qimage = ImageQt(item[0])
            img.setPixmap(QPixmap.fromImage(qimage))
            lay.addWidget(img)
        self.layout().addWidget(scroll, 0, 0, 1,
self.layout().columnCount())
        self.setStyleSheet("QScrollArea{min-width:300 px; min-
height: 400px}")

# Класс для функциональности переключения между моделями
class StringBox(QSpinBox):
    def __init__(self):
        super().__init__()

        strings = ["Model 1", "Model 2"]
        self.setStrings(strings)

    def setStrings(self, strings):
        strings = list(strings)
        self._strings = tuple(strings)
        self._values = dict(zip(strings, range(len(strings))))
        self.setRange(0, len(strings) - 1)

    def textFromValue(self, value):
        return self._strings[value]

app = QtWidgets.QApplication(sys.argv)
window = MainWindow()
```

```
window.show()
app.exec_()
```