

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К курсовому проекту

на тему

Самообучающийся алгоритм на основе нейронных сетей

Студент

гр. 053501
Кононович С.В.

Руководитель

ассистент кафедры информатики
Удовин И.А.

Минск 2020

Учреждение образования
«Белорусский государственный университет информатики и
радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий кафедрой
Информатики

(подпись)
Волорова Н.А. 2020 г.

ЗАДАНИЕ
по курсовому проекту

Студенту Кононовичу Станиславу Владимировичу

1. Тема работы Самообучающийся алгоритм на основе нейронных сетей

2. Срок сдачи студентом законченной работы 31.05.2021 г.

3. Исходные данные к работе Операционная система Windows. Язык программирования C++.

4. Содержание расчётно-пояснительной записки (перечень вопросов, которые подлежат разработке)

Введение. 1. Анализ предметной области. 2. Разработка программного средства. 3. Тестирование и проверка работоспособности. Заключение. Список использованных источников.

5. Консультант по курсовой работе Удовин И.А.

6. Дата выдачи задания 01.02.2021 г.

7. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и процентом от общего объёма работы):

раздел 1, Введение к 28.02.2020г. – 10 % готовности работы;

раздел 2 к 15.03.2020г. – 30 % готовности работы;

раздел 3 к 15.04.2020г. – 60 % готовности работы;

раздел 4 к 10.05.2020г. – 80 % готовности работы;

Заключение к 20.05.2020г. – 90 % готовности работы;

Оформление пояснительной записки и графического материала к 24.05.2020г. – 100 % готовности работы.

Защита курсового проекта с 31.05.2020 г. по 01.06.2020 г.

РУКОВОДИТЕЛЬ _____ Удовин И.А.
(подпись)

Задание принял к исполнению Кононович С.В. _____ 01.02.2020 г.
(дата и подпись студента)

ОГЛАВЛЕНИЕ

Введение	5
1. Анализ предметной области	6
1.1. Нейронные сети и матричные операции	6
1.2. Применение нейронной сети в распознавании изображений ...	7
1.2.1. Теория	7
1.2.2. Практика	7
1.3. Как нейросеть решает задачи по распознаванию образов	8
1.4. Выбор инструментария	8
1.5. Постановка задачи.....	9
2. Разработка программного средства.....	9
2.1. Интерфейс.....	9
2.2. Структура проекта.....	10
2.3. Реализация Image Parser	11
2.4. Реализация нейронной сети	11
2.5. Реализация графического интерфейса	13
3. Тестирование и проверка работоспособности	15
3.1. Проверка сети	15
3.2. Тест GUI 1.....	16
3.3. Тест GUI 2.....	17
3.4. Тест GUI 3.....	17
Заключение	18
Список источников.....	19

ВВЕДЕНИЕ

В связи с бурным развитием сферы информационных технологий в 2000-х годах произошел молниеносный рост вычислительных технологий, что привело к революции в сфере машинного обучения. Выделилась совершенно новая ветвь развития машинного обучения, которая до недавнего времени без огромных вычислительных мощностей считалась бесполезной и бесперспективной — нейронные сети. С течением времени люди научились разрабатывать и обучать самые различные архитектуры нейронных сетей, которые решают абсолютно разные типы задач: от распознавания лиц до игры в го с результатами, которые на голову превосходят человеческие.

Современные нейронные сети способны решать целый класс задач, в которых поиск наиболее оптимального решения нельзя четко алгоритмизировать. К примеру, нахождение оптимальной стратегии в компьютерных играх. Алгоритм, написанный человеком, может не учитывать всех факторов и совокупностей, влияющих на результат в игре, в отличие от гибких моделей нейронных сетей, которые методом проб и ошибок способны обучиться и находить пути решения задач, о которых люди могли даже и не подозревать.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Нейронные сети и матричные операции

Нейронная сеть — это последовательность нейронов, соединенных между собой синапсами. Структура нейронной сети пришла в мир программирования прямо из биологии. Благодаря такой структуре, машина обретает способность анализировать и даже запоминать различную информацию. Нейронные сети также способны не только анализировать входящую информацию, но и воспроизводить ее из своей памяти. Другими словами, нейросеть — это машинная интерпретация мозга человека, в котором находятся миллионы нейронов передающих информацию в виде электрических импульсов.

Нейрон — это вычислительная единица, которая получает информацию, производит над ней простые вычисления и передает ее дальше. Они делятся на три основных типа: входной, скрытый и выходной. Также есть нейрон смещения и контекстный нейрон. В том случае, когда нейросеть состоит из большого количества нейронов, вводят термин слоя. Соответственно, есть входной слой, который получает информацию, n скрытых слоев (обычно их не больше 3), которые ее обрабатывают и выходной слой, который выводит результат. У каждого из нейронов есть 2 основных параметра: входные данные (input data) и выходные данные (output data). В случае входного нейрона: $\text{input} = \text{output}$. В остальных, в поле input попадает суммарная информация всех нейронов с предыдущего слоя, после чего, она нормализуется, с помощью функции активации и попадает в поле output.

Функция активации — это способ нормализации входных данных и внесения нелинейности в модель. Без функции активации теряется смысл моделей нейронных сетей с большим количеством скрытых слоев, ведь все операции можно было бы заменить одним слоем с некоторым количеством нейронов.

Функция потерь (целевая функция) — функция, значение которой отражает успешность выполнения моделью поставленной задачи на некотором наборе данных. В общем, задачей всего машинного обучения является поиск подходящей функции и ее оптимизации (поиска глобального или близкого к глобальному минимума/максимума).

Нейронную сеть можно представить в виде математической модели, состоящей из линейных операций над матрицами. Слой входных нейронов будет представлен в виде вектора-строки параметров, которые передаются на

вход в нейронную сеть для получения предсказания. Каждый скрытый слой будет представлять собой матрицу $N \times M$, где каждый из M столбцов будет хранить в себе веса очередного нейрона в слое, состоящего из N нейронов. Выходной слой будет строиться по аналогии со скрытыми слоями, а нейрон смещений будет являться вектором-столбцов, устанавливающим, на какое значение необходимо сместить итоговый результат для достижения наиболее оптимального результата.

Последовательно перемножая матрицы, из которых состоит модель, и применяя функцию активации на каждом слое, мы получим итоговый результат предсказания модели, который будет в точности таким же, как если бы мы хранили нейросеть как наборы вершин и проводили бы операции на ней как обход графа. Поэтому данный метод является наиболее оптимальным для хранения нейронной сети в памяти компьютера.

1.2. Применение нейронной сети в распознавании изображений

Работа с изображениями — важная сфера применения технологий Deep Learning. Глобально все изображения со всех камер мира составляют библиотеку неструктурированных данных. Задействовав нейросети, машинное обучение и искусственный интеллект, эти данные структурируют и используют для выполнения различных задач: бытовых, социальных, профессиональных и государственных, в частности, обеспечения безопасности.

1.2.1. Теория

Основой всех архитектур для видеонаблюдения является анализ, первой фазой которого будет распознавание изображения (объекта). Затем искусственный интеллект с помощью машинного обучения распознает действия и классифицирует их.

Для того чтобы распознать изображение, нейронная сеть должна быть прежде обучена на данных. Это очень похоже на нейронные связи в человеческом мозге — мы обладаем определенными знаниями, видим объект, анализируем его и идентифицируем.

Нейросети требовательны к размеру и качеству датасета, на котором она будет обучаться. Датасет можно загрузить из открытых источников или собрать самостоятельно

1.2.2. Практика

На практике означает, что до определённого предела чем больше скрытых слоев в нейронной сети, тем точнее будет распознано

изображение. Как это реализуется?

Картинка разбивается на маленькие участки, вплоть до нескольких пикселей, каждый из которых будет входным нейроном. С помощью синапсов сигналы передаются от одного слоя к другому. Во время этого процесса сотни тысяч нейронов с миллионами параметров сравнивают полученные сигналы с уже обработанными данными.

Проще говоря, если мы просим машину распознать фотографию кошки, мы разобьем фото на маленькие кусочки и будем сравнивать эти слои с миллионами уже имеющихся изображений кошек, значения признаков которых сеть выучила.

В какой-то момент увеличение числа слоёв приводит к просто запоминанию выборки, а не обучению. Далее - за счёт хитрых архитектур.

1.3. Как нейросеть решает задачи по распознаванию образов

Нейронная сеть для распознавания изображений – это, пожалуй, наиболее популярный способ применения НС. При этом вне зависимости от особенностей решаемых задач, она работает по этапам, наиболее важные среди которых рассмотрим ниже.

В качестве распознаваемых образов могут выступать самые разные объекты, включая изображения, рукописный или печатный текст, звуки и многое другое. При обучении сети ей предлагаются различные образцы с меткой того, к какому именно типу их можно отнести. В качестве образца применяется вектор значений признаков, а совокупность признаков в этих условиях должна позволить однозначно определить, с каким классом образов имеет дело НС.

Важно при обучении научить сеть определять не только достаточное количество и значения признаков, чтобы выдавать хорошую точность на новых изображениях, но и не переобучиться, то есть, излишне не «подстроиться» под обучающую выборку из изображений. После завершения правильного обучения НС должна уметь определять образы (тех же классов), с которыми она не имела дела в процессе обучения.

Важно учитывать, что исходные данные для нейросети должны быть однозначны и непротиворечивы, чтобы не возникали ситуации, когда НС будет выдавать высокие вероятности принадлежности одного объекта к нескольким классам.

1.4. Выбор инструментария

Для реализации курсовой работы была выбрана интегрированная среда разработки CLion для разработки на языке C++. И был использован Qt Creator для создания графической оболочки.

CLion упрощает работу с C++, позволяя мне сосредоточиться на решении проблем. Также в данной среде разработки есть удобный Just-In-Time отладчик кода, позволяющий легко обнаружить ошибки в коде.

Qt Creator позволил создать GUI (graphical user interface) для взаимодействия с нейронной сетью в кратчайшие сроки. Благодаря удобным инструментам, например QWidget и QPaint, интерфейс получился привлекательным для конечного пользователя.

C++ — компилируемый, статически типизированный язык программирования общего назначения. Поддерживает такие парадигмы программирования, как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. Был выбран из-за его высокой производительности, требующейся для нейронной сети.

Git — распределённая система контроля версий.

GitHub — веб-сервис для хостинга IT-проектов и их совместной разработки.

1.5. Постановка задачи

В задачу курсовой работы входит разработка программного средства на языке C++, которое:

- Предоставляет возможность нарисовать число
- Стереть его
- Получить нарисованное число в текстовом виде

2. РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

2.1. Интерфейс

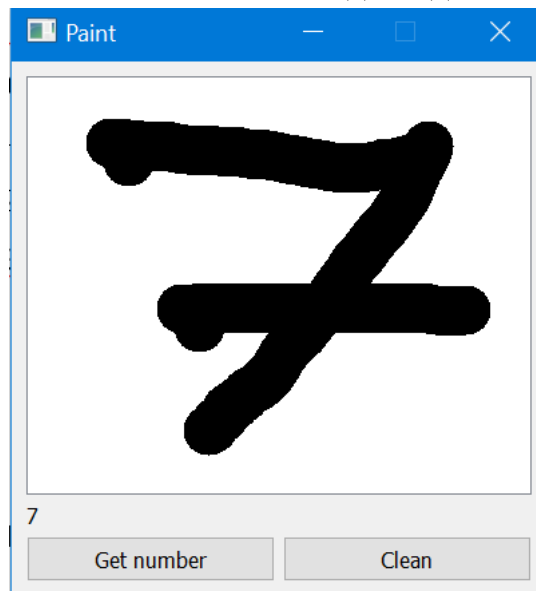
Для создания легкого и приятного в использовании интерфейса приложения были выделены следующие критерии:

- Интерфейс должен быть максимально функциональным, чтобы пользователь в нём не запутался
- Интерфейс должен быть удобен в использовании, чтобы пользователь мог легко начать использование нейронной сети
- Интерфейс должен быть приятным на вид и отвечать всем стандартам разработки современных десктопных приложений

Пользовательский интерфейс данного программного средства представляет собой 1 одно окно, которое было создано при помощи Qt Creator.

Пользовательские формы содержат в себе как и стандартные элементы управления QWidget, так и созданные на основе стандартных, как, например, GetNumber, унаследованная от класса QPushButton и созданная для улучшения интерфейса в силу отсутствия настраиваемости элементов, созданных при помощи элемента управления QPushButton. Окно содержит:

- graphicsView – поле для рисования
- Clean – кнопка, которая нужна для очистки поля рисования
- getNumber – кнопка, активирующая нейронную сеть текущим изображением на graphicsView
- outPut – QLabel, которое выводит результат выполнения нейронной сети в тестовом виде под полем для рисования



2.2. Структура проекта

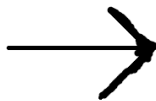
Проект состоит из трёх независимых программ:

- Image_Parser – нужен для конвертирования изображений в понятный для нейронной сети текстовый формат
- NeuralNetwork – программа для обучения нейронной сети на данных, полученных с помощью Image_Parser. Полученные веса сохраняются в weights.txt для дальнейшего использования
- NumberRecognizer – программа, которая использует веса, полученные во время обучения, чтобы преобразовать данные пользователя (картинку) в число (текстовый формат). Также содержит в себе GUI.

2.3. Реализация Image Parser

Содержит лишь две функции:

- `int main()` в `main.cpp`, которая вызывает метод `getDots` из `parser.cpp` и сохраняет полученные значения.
- `std::vector<double> getDots(const QString& filename, int width, int height)` в `parser.cpp`. Принимает имя файла и разрешение, в котором его нужно обработать. Далее считывает значения пикселей, и возвращает вектор значений, степени чёрного в точке (изменяется от 0 до 1).



```
lib.txt – Блокнот
Файл  Правка  Формат  Вид  Справка
0 0 0 0 0 0 0 0 0 0 0.282353 0.87451 1 0.87451 0.376471 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0.282353 0.87451 1 1 1 1 1 1 0.87451 0.376471 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0.282353 0.937255 1 1 1 1 1 1 1 1 0.968627 0.376471 0 0 0 0 0 0 0
0 0 0 0 0 0 0.905882 1 1 1 1 1 1 0.780392 1 1 1 1 1 0.968627 0.407843 0 0 0 0 0 0
0 0 0 0 0 0.282353 1 1 1 0.592157 0 0 0 0 0 0.470588 1 1 1 1 0.470588 0 0 0 0 0 0
0 0 0 0 0 0.905882 1 1 1 0 0 0 0 0 0.470588 1 1 1 0.968627 0.376471 0 0 0 0 0 0
0 0 0 0 0 1 1 1 0.592157 0 0 0 0 0 0 0.0313725 0.968627 1 1 0.968627 0 0 0 0 0
0 0 0 0 0.282353 1 1 1 0 0 0 0 0 0 0 0.470588 1 1 1 0 0 0 0 0 0
0 0 0 0 0.905882 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0
0 0 0 0 1 1 1 0.592157 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0
0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0
0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0
0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0.282353 1 1 1 0 0 0 0
0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0.905882 1 1 1 0 0 0 0
0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 0.592157 0 0 0 0
0 0 0 0.282353 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0
0 0 0 0.905882 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0
0 0 0 1 1 1 0.74902 0 0 0 0 0 0 0 0 0 0.282353 1 1 1 0 0 0 0
0 0 0 0.968627 1 1 0.968627 0 0 0 0 0 0 0 0 0.905882 1 1 1 0 0 0 0
0 0 0 0.470588 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 0.592157 0 0 0 0 0
0 0 0 0 1 1 1 0.376471 0 0 0 0 0 0 0 0.282353 1 1 1 0 0 0 0 0
0 0 0 0.968627 1 1 0.968627 0 0 0 0 0 0 0.905882 1 1 1 0 0 0 0 0
0 0 0 0.470588 1 1 1 0.87451 0.376471 0 0 0 0 0.282353 1 1 1 0.592157 0 0 0 0 0
0 0 0 0.968627 1 1 1 0.968627 0.376471 0 0 0.282353 0.937255 1 1 1 0 0 0 0 0
0 0 0 0 0.470588 1 1 1 1 0.968627 0.529412 0.87451 1 1 1 1 0.592157 0 0 0 0 0
0 0 0 0 0 0.470588 1 1 1 1 1 1 1 0.592157 0 0 0 0 0 0 0
0 0 0 0 0 0 0.470588 1 1 1 1 1 1 0.592157 0 0 0 0 0 0 0
0
<----->
```

2.4. Реализация нейронной сети

Сама нейронная сеть, а точнее класс для её создания, находится в

network.cpp и содержит:

`struct Neuron` - структура нейрона:

- `double error` — поле ошибки
- `double value` — поле значения
- `void act()` - функция активации нейрона

`class Network` — класс для создания нейронной сети

- `int layers` — поле, в котором хранится значение количества слоёв нейронной сети
- `Neuron** neurons` — двумерный массив, в котором содержатся нейроны
- `double*** weights` — трёхмерный массив, в котором хранятся веса
Характеристика, определяющая влияние нейрона прошлого слоя на нейрон текущего слоя
- `int* size` — массив, в котором находятся количество нейронов в слоях
- `int threadsNum` — поле, в котором хранится число потоков этого компьютера. Используется дальше для применения многопоточности
- `double sigmaDerivative(double x)` — метод, берущий производную от сигмоидной функции. Сигмоидная функция нужна для того, чтобы “ужать” принимаемые в нейрон значения до диапазона от 0 до 1
- `double predict(double x)` — метод предсказания, определяет, загорится ли нейрон при таких-то входных данных
- `void setLayersNotStudy(int n, int* p, std::string filename)` — метод, принимает размеры сети и заполняет её значениями (весами), хранящимися в указанном файле
- `void setLayers(int n, int* p)` — метод, принимает размеры нейронной сети и заполняет её случайными значениями
- `void setRandomInput()` — метод, генерирует случайный вход. Использовался для отладки сети во время разработки
- `void setInput(const double p[])` — метод, который принимает входные данные для обработки нейронной сетью. Входные данные – это степень чёрного в пикселе в диапазоне от 0 до 1 (1 – полностью чёрный, 0 – белый)
- `void show()` — метод для отладки нейронной сети. Выводит важную информацию о её внутреннем устройстве. Использовался для отладки во время разработки
- `void LayersCleaner(int LayerNumber, int start, int stop)` — метод для очистки слоя. Значения start и stop нужны для многопоточности, чтобы разные потоки не чистили одни и те же ресурсы, вызывая сбой программы
- `void ForwardFeeder(int LayerNumber, int start, int stop)` — вспомогательный метод для ForwardFeed. Нужна для распределения затратных манипуляций разным потокам
- `double ForwardFeed()` — метод для передачи значений через слои нейронной

сети. Возвращает обработанное значение (результат выполнения нейронной сети)

- `double ForwardFeed(const std::string& param)` — тот же `ForwardFeed`, но выводит информацию о степени активации нейронов выходного слоя на консоль. Использовался для отладки во время разработки
- `void ErrorCounter(int LayerNumber, int start, int stop, double prediction, double rightResult, double lr)` — метод, который считает ошибку для каждого нейрона
- `void WeightsUpdater(int start, int stop, int LayerNum, int lr)` — метод, обновляющий веса
- `void BackPropagation(double prediction, double rightResult, double lr)` — метод, совершающий обратное распространение ошибки. Метод обратного распространения ошибки (англ. *backpropagation*) — метод вычисления градиента, который используется при обновлении весов многослойного перцептрона. Основная идея этого метода состоит в распространении сигналов ошибки от выходов сети к её входам, в направлении обратном прямому распространению сигналов в обычном режиме работы.
- `bool SaveWeights(const std::string& fileName = "weights.txt")` — метод, сохраняющий веса, полученные во время обучения, в текстовый файл для дальнейшего использования
- `int main()` в `main.cpp` (в директории с нейронной сетью) — функция, которая обучает нейронную сеть, «скармливая» ей обработанные в `Image Parser` .png файлы. Далее тестирует нейронную сеть на предмет эффективности и корректности

2.5. Реализация графического интерфейса

Создано при помощи Qt Creator. Включает в себя три файла. `main.cpp` — для запуска виджета, который был сконструирован в `paint.cpp`. `paintscene.cpp` содержит `class paintScene` для рисования. Остановимся подробнее на `paint.cpp`

- `explicit Paint(QWidget *parent = 0)` — конструктор, в котором устанавливается `ui`, значения кнопок и `QLabel`. Также происходит connect события нажатия кнопки и метода для его обработки
- `void resizeEvent(QResizeEvent * event)` - Переопределяем событие изменения размера окна для пересчёта размеров графической сцены
- `paintScene *scene` - Объявляем кастомную графическую сцену
- `Ui::Paint *ui` — Объявляем `ui`
- `QTimer *timer` - Определяем таймер для подготовки актуальных размеров графической сцены
- `void getNumber()` — вызывается при нажатии соответствующей кнопки. Сохраняет данные графической сцены в png, далее png преобразовывается в

набор точек, понятных нейронной сети с помощью функции getDots из Image Parser. Потом конструируется нейронная сеть с архитектурой 784*16*16*10 и заполняется весами из NeuralNetwork. Далее нейронная сеть получает входные значения и возвращает результат. Который оказывается в QLabel прямо под графической сценой

- **void clean()** – вызывается при нажатии соответствующей кнопки. Очищает графическую сцену

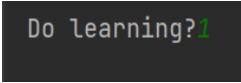
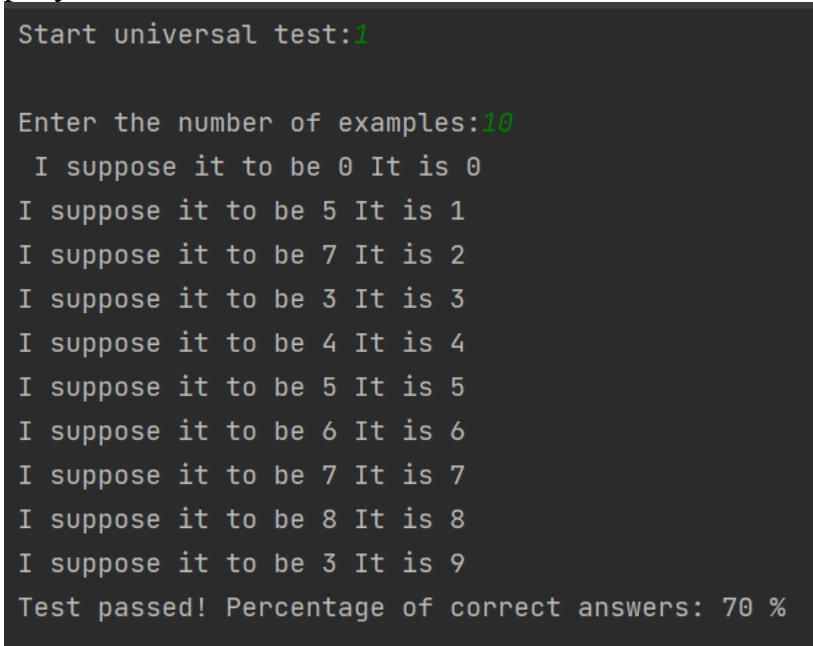
Исходный код paint.ui:

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
  <class>Paint</class>
  <widget class="QWidget" name="Paint">
    <property name="geometry">
      <rect>
        <x>0</x>
        <y>0</y>
        <width>400</width>
        <height>300</height>
      </rect>
    </property>
    <property name="windowTitle">
      <string>Paint</string>
    </property>
    <layout class="QGridLayout" name="gridLayout">
      <item row="2" column="0">
        <widget class="QPushButton" name="getNumber">
          <property name="text">
            <string>Get Number</string>
          </property>
        </widget>
      </item>
      <item row="2" column="1">
        <widget class="QPushButton" name="clean">
          <property name="text">
            <string>Clean</string>
          </property>
        </widget>
      </item>
      <item row="0" column="0" colspan="2">
        <widget class="QGraphicsView" name="graphicsView"/>
      </item>
      <item row="1" column="0" colspan="2">
        <widget class="QLabel" name="outPut">
          <property name="text">
            <string>Number</string>
          </property>
        </widget>
      </item>
    </layout>
  </widget>
  <layoutdefault spacing="6" margin="11"/>
  <resources/>
  <connections/>
</ui>
```

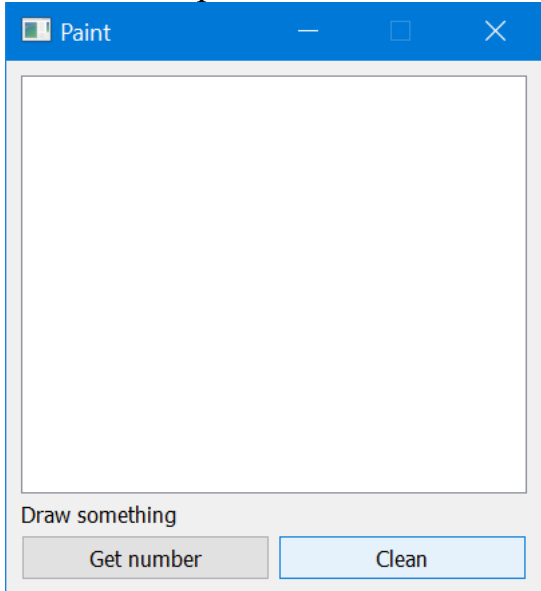
3. ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ

В ходе тестирования были проверены ключевые функции, из которых следует работоспособность всего программного средства.

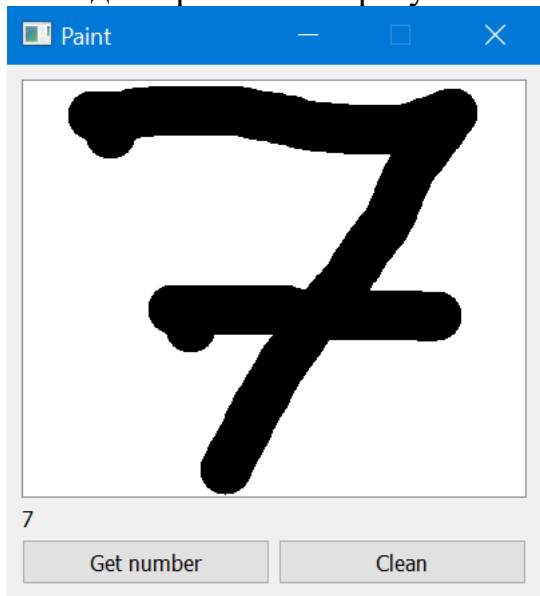
3.1. Проверка сети

Тестовая ситуация	<p>Нейронная сеть обучена на 60 образцах цифр. Каждая цифра встречается 6 раз. Далее она тестируется на 10 образцах разных цифр</p> 
Ожидаемый результат	Результат тестирования в процентах не меньше 70. Ведь нейронная сеть была обучена на крайне малом наборе.
Фактический результат	<p>Нейронная сеть показала удовлетворительные результаты</p> 

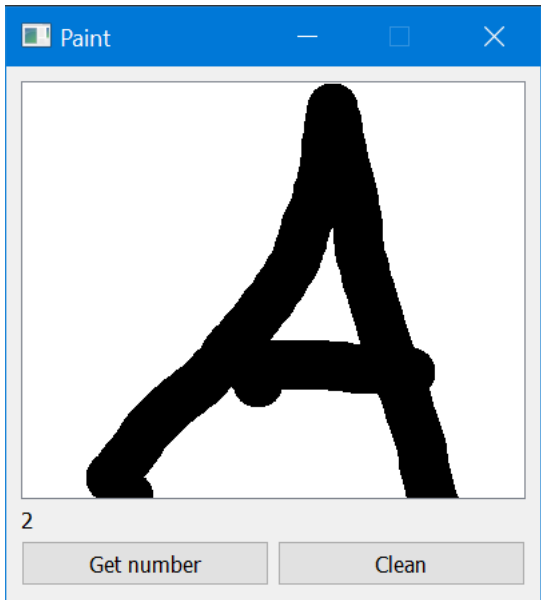
3.2. Тест GUI 1

Тестовая ситуация	Попытка очистить пустую графическую сцену
Ожидаемый результат	Бездействие
Фактический результат	Ничего не произошло 

3.3. Тест GUI 2

Тестовая ситуация	Ввод числа 7 и нажатие кнопки Get number
Ожидаемый результат	Вывод числа 7 под графическим редактором
Фактический результат	<p>Выведен правильный результат</p> 

3.4. Тест GUI 3

Тестовая ситуация	Попытка распознать ненастоящее число
Ожидаемый результат	Вывод случайного результата под графическим редактором
Фактический результат	<p>На выходе получили случайное число (в этот раз 2)</p> 

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы было разработано приложение, корректно справляющееся с процессом обучения нейронной сети и сохранением результата обучения для дальнейшего использования. Было разработано приложения для обработки изображений в понятный для нейросети формат. Далее было разработан графический интерфейс взаимодействия с сетью, удобный для пользователя. Нейронная сеть показала отличные результаты, хоть и была обучена на малой выборке. Благодаря продуманной архитектуре сети она может быть быстро адаптирована и обучена для схожих задач, например распознавания букв латинского алфавита.

СПИСОК ИСТОЧНИКОВ

1. Wikipedia: <https://ru.wikipedia.org/wiki/>
2. Qt Documentation: <https://doc.qt.io/>
3. Proglib: <https://proglib.io/p/neural-network-course>
4. Cplusplus: <https://www.cplusplus.com/>