Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

Розробка нейронної мережі для розпізнавання рукописних графічних об’єктів

Текстова частина до курсової роботи
За спеціальністю «Інженерія програмного забезпечення» 121

Керівник курсової роботи
к.т.н., доц. Жежерун О. П.

_____
(підпис)
“____” _____ 2020 р.

Виконав студент 3-го курсу
“____” _____ 2020 р.

Київ 2020

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри мультимедійних систем,

доцент, к.т.н.

_____ О. П. Жежерун

(підпис)

“____”_____ 2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту _____Білокіню Данилу_____

__3-го__ курсу факультету інформатики

ТЕМА: Розробка нейронної мережі для розпізнавання рукописних графічних об'єктів.

Вихідні дані:

- Застосунок для розпізнавання рукописних цифр

Зміст ТЧ до курсової роботи:

Вступ
1. Аналіз предметної області. Постановка завдання курсової роботи
2. Теоретичні відомості
3. Опис реалізації програмного продукту
Висновки
Список джерел
Додатки (за необхідністю)

Дата видачі “___” _____ 2020 р.

Керівник _____ Завдання отримано _____

Календарний план виконання курсової роботи

Тема: Розробка нейронної мережі для розпізнавання графічних об’єктів

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	Жовтень-листопад 2019р.	
2.	Огляд літератури за темою роботи	Листопад-грудень 2019р.	
3.	Аналіз сучасних методів	Січень 2020р.	
4.	Реалізація нейронної мережі	Лютий 2020р.	
5.	Реалізація графічного інтерфейсу застосунку та тестування	Квітень 2020р.	
6.	Написання текстової частини.	Квітень 2020р.	
7.	Перегляд курсової роботи науковим керівником	Травень 2020р.	
9.	Перегляд змісту роботи керівником	07.05. 2020	
10.	Внесення змін до курсової роботи відповідно до зауважень наукового керівника	09.05. 2020	
11.	Створення презентації	10.05.2020	
12.	Захист курсової роботи	18.05.2020	

Студент Білокінь Д.Т. _____

Керівник Жежерун О.П. _____

“_____”_____ 2020 р.

Зміст

Анотація

Робота присвячена вивченню роботи нейронної мережі та побудова застосунку з користувацьким інтерфейсом з можливістю малювання для розпізнавання рукописних графічних об’єктів із зображення, зокрема цифр, на основі отриманих знань.

У застосунку, розробленому під час виконання цієї роботи, використано нейронну мережу на основі моделі перцептрон.

Реалізація на мові Java з використанням стандартних бібліотек.

Вступ

Машинне навчання – це область штучного інтелекту, в якій використовуються статичні методи, щоб дати змогу комп'ютерній програмі навчатися за допомогою конкретних даних, не будучи явно запрограмованою. Машинне навчання розвивається кожного дня та набуває все більшої популярності. Зараз його використовують в дуже багатьох сферах людської діяльності. Наприклад, онлайн-перекладачі, розпізнавання тексту, пошук інформації, фільтрація поштових скриньок, реклама на основі переглядів користувачів, чат-боти, забезпечення для керування автомобілями(система автопілоту), тощо.

Справжнім успіхом в машинному навчанні стали голосові помічники з можливістю розпізнавати мовлення, наприклад, Apple Siri. Саме з них почалось активне використання машинного навчання в повсякденному житті. Зараз нейронні мережі виконують і складніші завдання, такі як керування автомобілем.

Ця робота має на меті огляд нейронних мереж та більш детально зупинитись на мережі, яка моделює процеси людського сприйняття, а саме «перцептрон» (від латинського perception – сприйняття) та її використання для розпізнавання графічних об'єктів, та створення застосунку, який продемонструє роботу нейронної мережі на прикладі розпізнавання рукописних цифр.

Робота складається з трьох розділів.

В першому розділі розглянуто сучасний стан речей в розпізнаванні графічних об'єктів із зображення та розглянуті можливі існуючі аналоги розробки.

В другому розділі коротко описані різні підходи машинного навчання та детально розглянуто нейронні мережі, на прикладі мережі, яка використовує архітектуру «перцептрон». Також детально розглянуто алгоритми для навчання мережі, зокрема алгоритм зворотного розповсюдження.

В третьому розділі описано детальний процес розробки та тестування застосунку.

1. Аналіз предметної області. Постановка завдання курсової роботи

1.1. Аналіз сучасного стану питання

З часу перших винаходів в сфері розпізнавання об’єктів ця сфера значно просунулась вперед. Справжнім проривом на той час була розробка першої машини для перетворення друкованого тексту в машинну мову. Першою програмою для розпізнавання кирилиці була «AutoR», яка побачила світ в 1992 році. На поточний момент часу точність програм для розпізнавання друкованого тексту досягає 99%, абсолютної точності не може забезпечувати жоден застосунок. Точність розпізнавання рукописного друкованого тексту ще нижча, 80 - 90%. Ще більші проблеми наразі з розпізнаванням не друкованого тексту. В першу чергу проблеми з розпізнаванням рукописного друкованого тексту викликають такі фактори як, розташування тексту під кутом та особливості почерку, тому задача досі не вважається вирішеною на всі 100%. Проблеми розпізнавання рукописних символів досі залишаються темою активних обговорень та досліджень.

1.2. Огляд існуючих аналогів розробки

Наразі є дуже багато компаній та застосунків, які використовують технології розпізнавання тексту. Напевно, найвідомішою такою компанією є «ABBYY» та їх застосунок «FineReader». На даний момент застосунок розпізнає текст на 192 мовах світу і має вбудовано систему перевірки орфографії для 48 з них. На офіційному сайті «ABBYY FineReader» вказано, що за допомогою їх технологій можна розпізнати текст з файлу або за допомогою мобільного додатку отримати текст з камери смартфону.

ABBYY FineReader – «універсальне рішення для роботи з паперовими та PDF файлами будь-якого типу. Поєднання системи оптичного розпізнавання тексту (OCR — Optical Character Recognition) та інструменти для роботи з PDF документами дозволяє ефективно вирішувати ваші задачі.» [1]

Як ABBYY FineReader розпізнає текст? Спочатку він аналізує структуру. Розділяє документ на елементи такі як: таблиці, блоки тексту. Отримані елементи поділяються на слова, які в свою чергу поділяються на символи. Потім символи порівнюються зі зразками програми та будуються гіпотези щодо того який це символ може бути.

1.3. Постановка задачі

1) Спочатку потрібно підібрати базу з даними для тренування моделі. Задля отримання максимально релевантних даних потрібно добре вивчити всі особливості бази даних.
2) Створити нейронну мережу та навчити її на основі нашої бази.
3) Створення графічного інтерфейсу за допомогою якого користувач може намалювати цифру та побачити результат.
4) Тестування програми

2. Теоретичні відомості

2.1. Основні відомості та огляд підходів до машинного навчання

Машинне навчання – розділ штучного інтелекту, який вивчає методи побудови алгоритмів, які вміють навчатися.

Розглянемо найпопулярніші методи машинного навчання:

Навчання з вчителем

Найпопулярніший метод. Кожен прецедент представляє собою пару «об’єкт, відповідь». Потрібно знайти функціональну залежність відповідей від характеристик об’єкта та побудувати алгоритм, який приймає опис об’єкту, а на виході виводить відповідь.

Прикладами задач на навчання з вчителем є класифікація, регресія та прогнозування.

Навчання без вчителя

В даному методі правильні відповіді не надаються, потрібно знайти залежності між об’єктами.

Прикладами задач на навчання без вчителя є кластеризація, пошук асоційованих правил та скорочення розмірності.

Часткове навчання

Проміжний метод між навчання з вчителем та без. Кожен прецедент представляє собою пару «об’єкт, відповідь», але відповіді відомі тільки на частині прецедентів.

Прикладом задачі на часткове навчання є віднесення певної статті в рубрику, з умовою, що деякі статті вже віднесені в певні рубрики.

Трансдуктивне навчання

Відома кінцева вибірка прецедентів, потрібно по цим відомим даним зробити прогноз по відношенню до іншої вибірки даних – тестової. На відміну від стандартного методу тут не потрібно знаходити закономірність, тому що відомо, що нових даних не буде.

Навчання з підкріпленням

В даному випадку система запам’ятовує всі дані напам’ять, зберігає в пам’яті всі навчальні приклади. При класифікації нові приклади порівнюються зі старими за допомогою міри подібності, при чому старі приклади можуть видалятися, а на їх місце записуватися нові.

Прикладом навчання з підкріпленням є навчання роботів.

Описані вище алгоритми відносяться до класичних в машинному навчанні, але більш сучасними є нейронні мережі, тому що вони не запрограмовані на конкретну задачу.

Головною ідеєю нейронних мереж є імітування процесів обробки інформації в людському мозку. Цю ідею розвивали в 50-80-их роках XX століття Воррен Маккалох, Уолтер Пітс, Френк Розенблатт, Джон Хопфілд.

Щоб краще зрозуміти як працює нейронна мережа в комп’ютерній сфері, розглянемо як це працює на біологічному рівні в людському мозку, а саме як нейрони обмінюються інформацією з іншими. Як інформаційна система, нейрон має входи, які називаються синапсами, тіло(сому), яке реалізує обробку інформації, своєрідний процесор, та вихід – аксон. Сигнали, які утворюються в тілі нейрона, проходять через аксон на синапс наступного нейрона, після чого взаємодіє с сигналами, які прийшли з інших нейронів та визначає поведінку цього нейрона.

2.2. Нейронна мережа на основі архітектури «перцептрон»

2.2.1. Основі поняття

Найпершим хто запропонував прилад, який моделює людське сприйняття був Френк Розенблатт. В найпершому перцептроні в ролі аксонів виступали фотоелементи, а в ролі синапсів блоки електромеханічних комірок пам'яті. Ці комірки пам'яті з'єднувалися між собою випадковим чином. В 1957 році було остаточно змодельовано роботу перцептрона на комп'ютері IBM 704.

Для початку введемо поняття про складові частини перцептрона[4]:

Простий S-елемент – аналог синапсу, який виробляє сигнал при дії на нього будь-якого виду енергії, якщо сигнал вище вказаного порогу, то елемент видає вихідний сигнал дорівнює одиниці, в протилежному випадку нулеві.

Простий A-елемент – аналог соми, який видає вихідний сигнал, якщо сума всіх вхідних сигналів вище заданого порогу. Якщо сума більше заданого значення, то вихідний сигнал дорівнює одиниці, в протилежному випадку нулеві.

Простий R-елемент – аналог аксону, який видає сигнал, який дорівнює одиниці, якщо сума сигналів, що входять в елемент є строго більшою від нуля, В протилежному випаду, якщо сума сигналів, що входять, є меншою від нуля, то сигнал дорівнює одиниці зі знаком мінус. Якщо сума дорівнює рівно нулю, то і сигнал на виході дорівнює нулю.

Логічна відстань між елементами – число шарів між ними.

Перцептрон це доволі широке поняття, тому приведемо деякі поняття, які ввів Розенблатт[4]:

Перцептрон

Мережа, яка складається з усіх трьох складових елементів та взаємодіє з матрицею W(елементи якої w_{ij} –коефіцієнти ваги), яка задається послідовністю всіх попередніх станів активної мережі.

Перцептрон з послідовними зв'язками[4]

Мережа, в якої всі зв'язки, які починаються від елементів з логічною відстанню d від найближчого чутливого елемента, закінчуються на елементах з логічною відстанню d+1 від найближчого чутливого елемента.

Перцептрон з перехресними зв'язками

Система, в якій деякі зв'язки з'єднують один з одним елементи одного типу, які знаходяться на однаковій логічній відстані від чутливих елементів, причому всі інші зв'язки мають послідовний типу.

Простий перцептрон

Система, яка відповідає п'яти умовам:

- В мережі є тільки один реагуючий елемент, який зв'язаний зі всіма логічними елементами;
- Система представляє собою перцептрон, який має виключно послідовні зв'язки, які йдуть тільки від чутливих до логічних елементів та від логічних до реагуючих елементів;
- Ваги всіх зв'язків від чутливих до логічних елементів є сталими величинами;
- Час проходження для кожного зв'язку системи дорівнює нулю або фіксованій сталі t
- Всі активуючі функції мають вигляд:
  $U_i = f\left(a_i(t)\right)$, де $a_i(t)$ – сума всіх сигналів, які поступають на вхід елементу u_i.

2.2.2. Поняття нейрона

Так виглядає штучний нейрон в схематичному вигляді(Рисунок 2.1)[5]

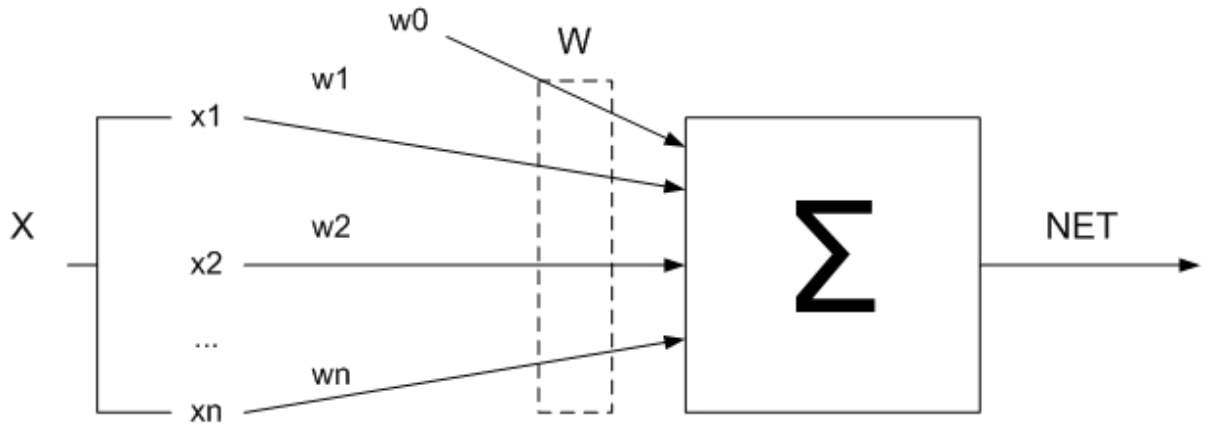

Рисунок 2.1

Штучний нейрон імітує поведінку біологічного нейрона. На вхід нейрона приходить кілька сигналів, кожен з яких є виходом іншого нейрона. Кожен вхід множиться на відповідну вагу, а потім всі добутки додаються, що визначає рівень активації нейрона.

$$OUT = \sum_{i=1}^{n} w_i \bullet x_i + w_0 ,$$

де w_0 – біас(нейрон зміщення);

w_i – вага і-го нейрона;

x_i – вихід і-го нейрона;

n – кількість нейронів, що входять в даний.

2.2.3. Функція активації

Отриманий сигнал як правило обробляється функцією активації(Рисунок 2.2)[5]

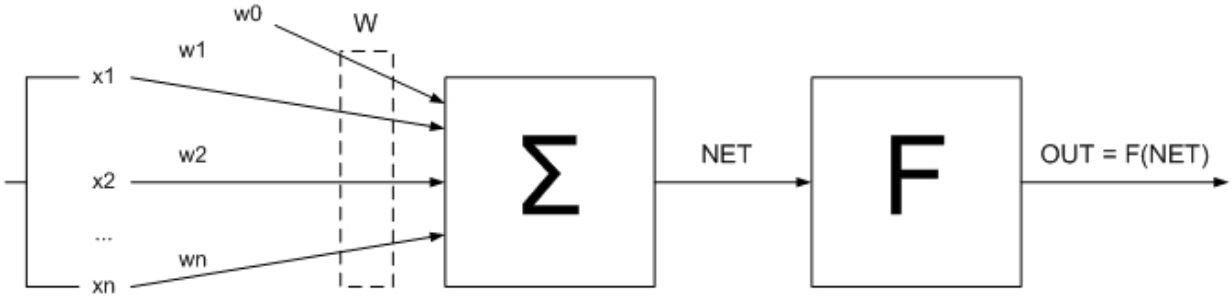

Рисунок 2.2

Функція активації потрібна для того, щоб вихідне значення завжди було в межах деяких значень. Частіше всього для цього використовується так звана «сигмоїдальна» функція. Математично вона виражається так:

$$sigmoid = \frac{1}{1 + e^{-OUT}}$$

Плюсом цієї функції є те, що вона має просту похідну та диференціюється на всій вісі абсцис. Графік виглядає так(Рисунок 2.3)[10]:

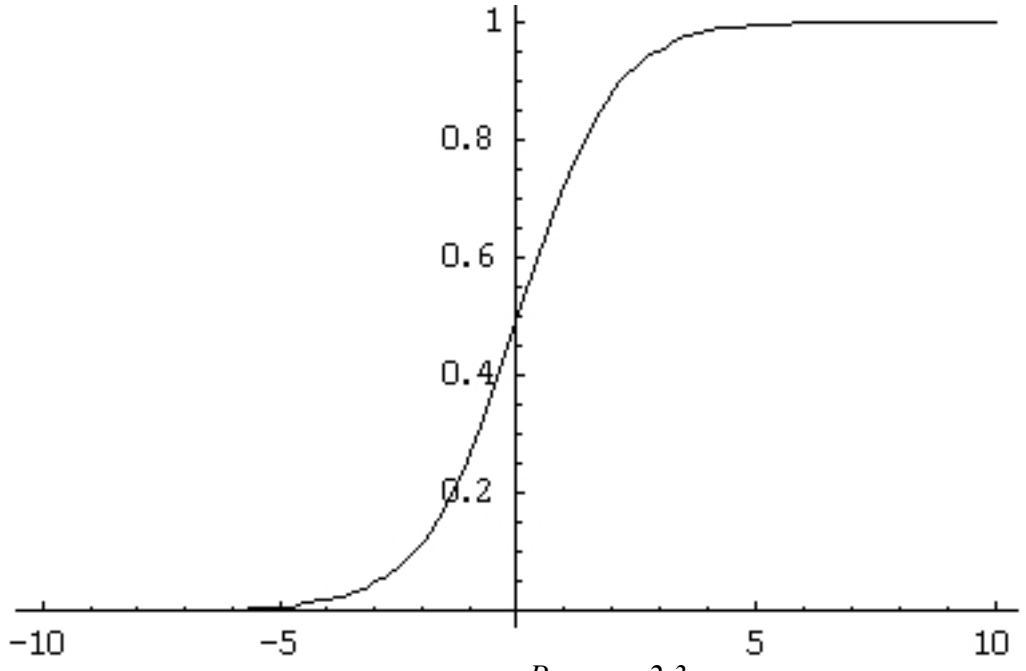

Рисунок 2.3

2.2.4. Алгоритми навчання

Класичним методом навчання є алгоритм корекції помилки. В цьому алгоритмі вага зв’язку змінюється тільки при умові, що реакція є неправильною, в такому випадку вона змінюється на одиницю з протилежним знаком від значення помилки.

Ми розглянемо алгоритм зворотного розповсюдження помилки

Розглянемо на прикладі нейронної мережі, яка складається з трьох шарів, яка має два входи та один вихід(Рисунок 2.4)[6]

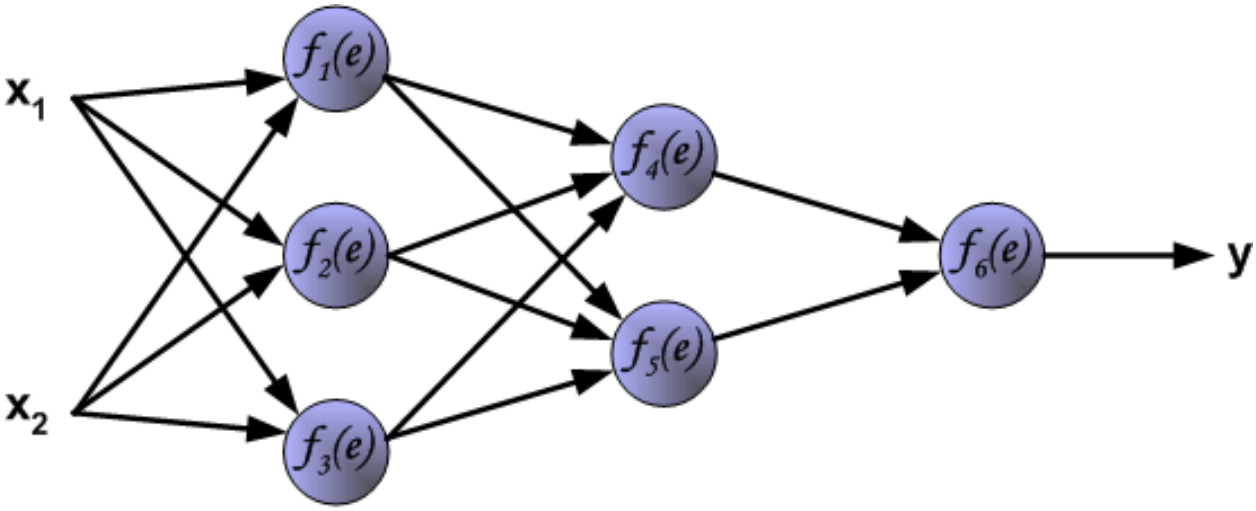

Рисунок 2.4

Щоб навчити нейронну мережу, ми маємо взяти дані для навчання. В нашому випадку це сигнали x_1, x_2 та очікуваний результат z. З кожною ітерацією вагові коефіцієнти підганяються з використанням нових тренувальних даних. В зміні вагових коефіцієнтів і полягає сенс алгоритму.

Згідно з формулами для знаходження вихідних сигналів та функції активації знаходимо значення вихідного нейрона(Рисунок 2.5) [6].

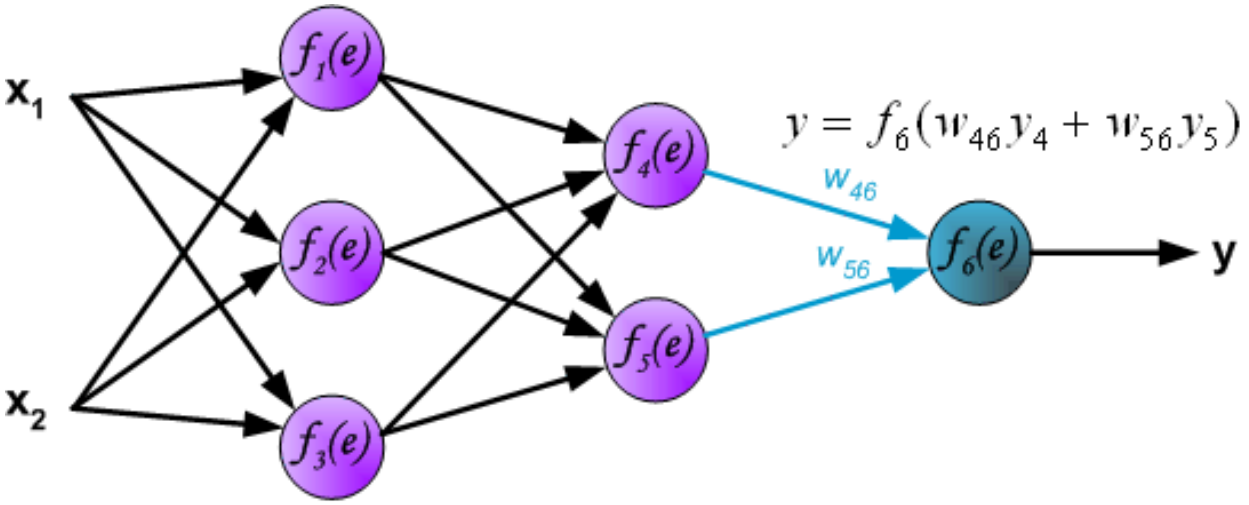

Рисунок 2.5

На наступному кроці алгоритму результат(y) порівнюється з очікуваним(z), різниця між цими значеннями(δ) називається помилкою вихідного шару(Рисунок 2.6) [6].

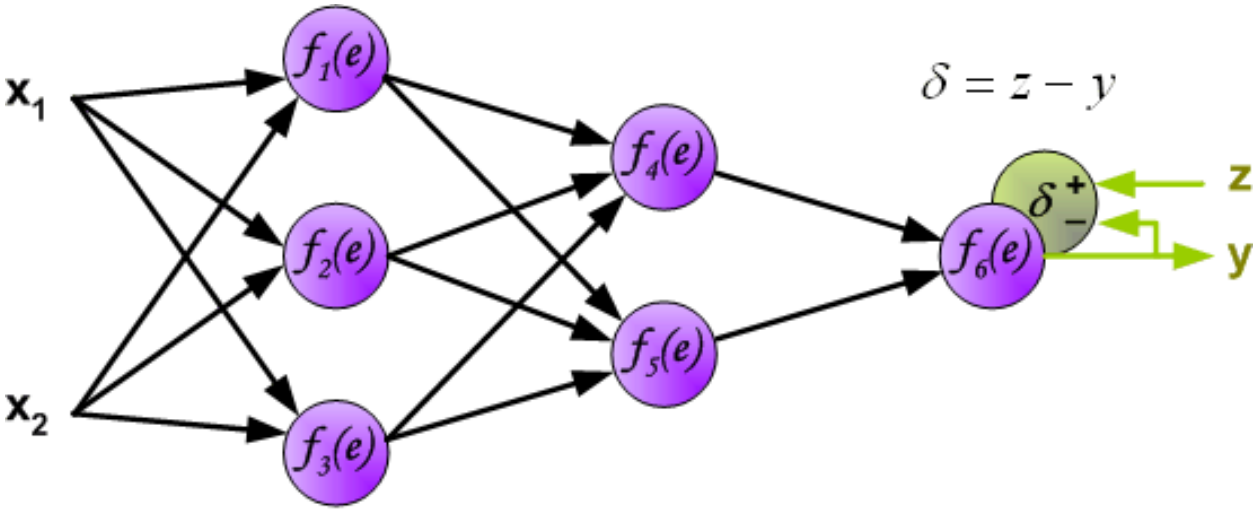

Рисунок 2.6

Ідея цього алгоритму полягає в розповсюдженні сигналу помилки(δ) в зворотному напрямку на всі нейрони, чиї сигнали були вхідними(Рисунок 2.7 та Рисунок 2.8) [6].

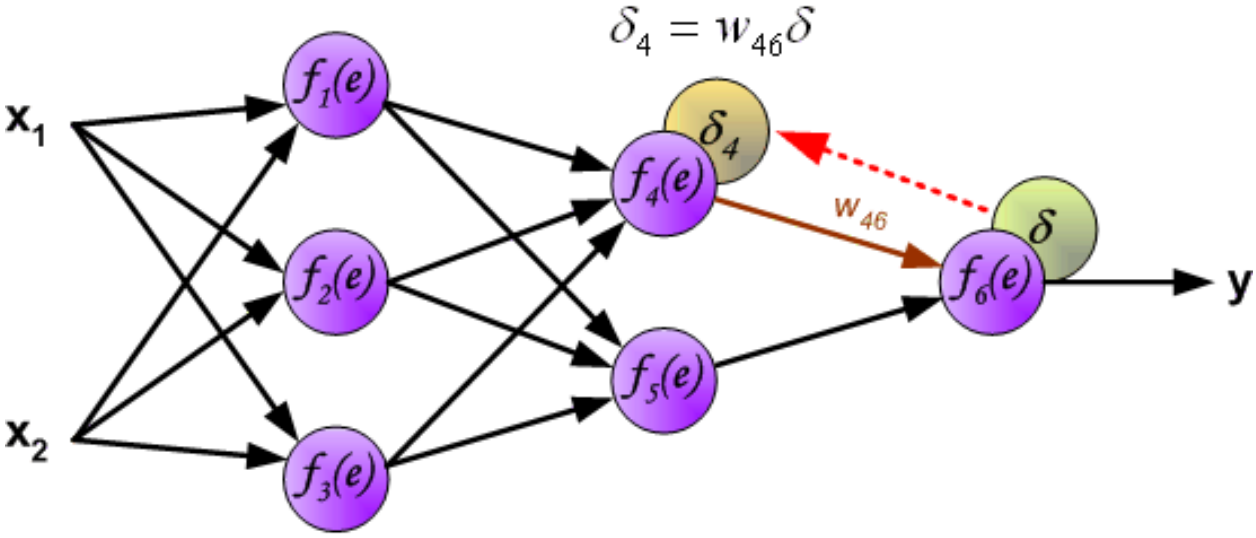

Рисунок 2.7

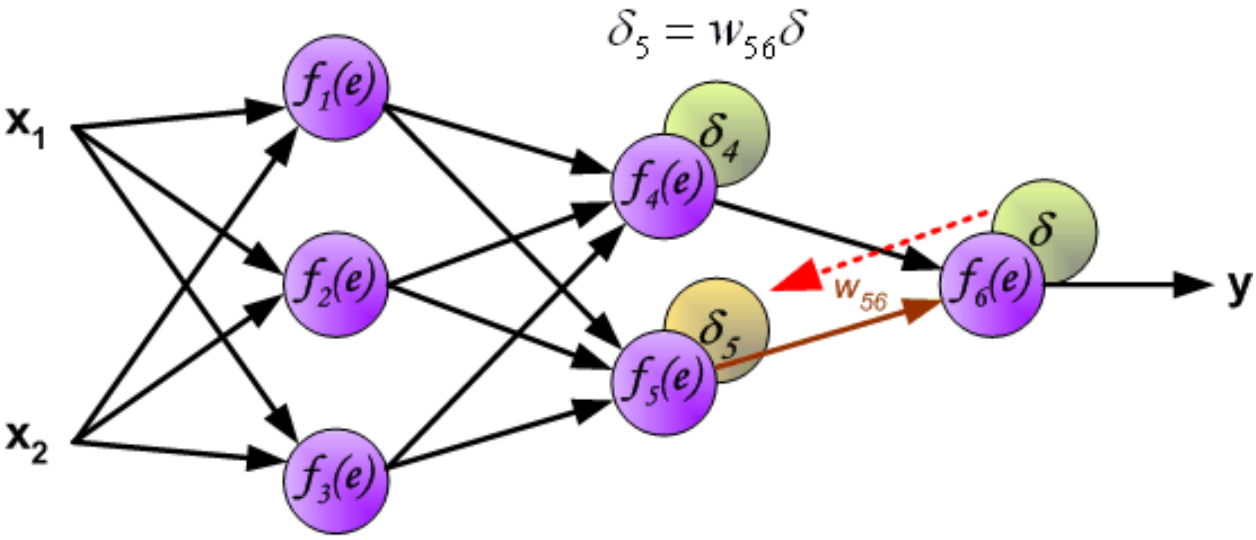

Рисунок 2.8

Вагові коефіцієнти залишаються тими, що і були. Якщо нейрон був вхідним для кількох нейронів, то значення сумуються(Рисунок 2.9) [6].

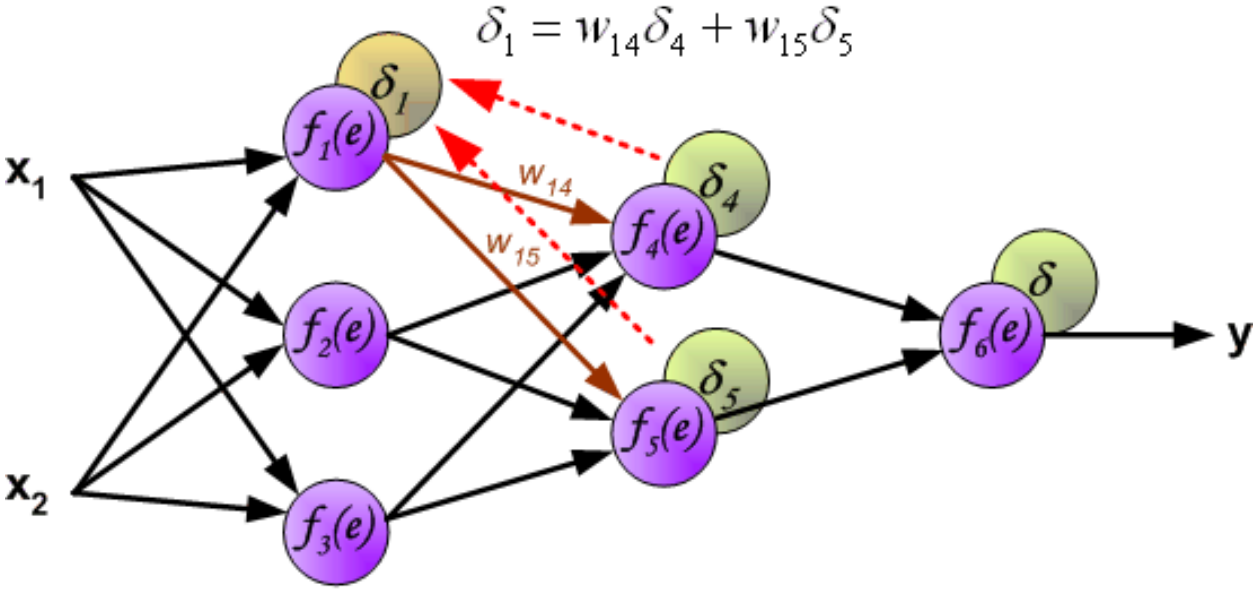

Рисунок 2.9

Коли ми знайшли величину помилки для кожного нейрона можна скорегувати вагові коефіцієнти кожного вузла вводу. Для корегування ваг використовується похідна від функції активації, а саме:

$$deactivation = activaion(x) \bullet (1 - activation(x))$$

Також у нас з'являється коефіцієнт η, який впливає на швидкість навчання(Рисунок 2.10) [6].

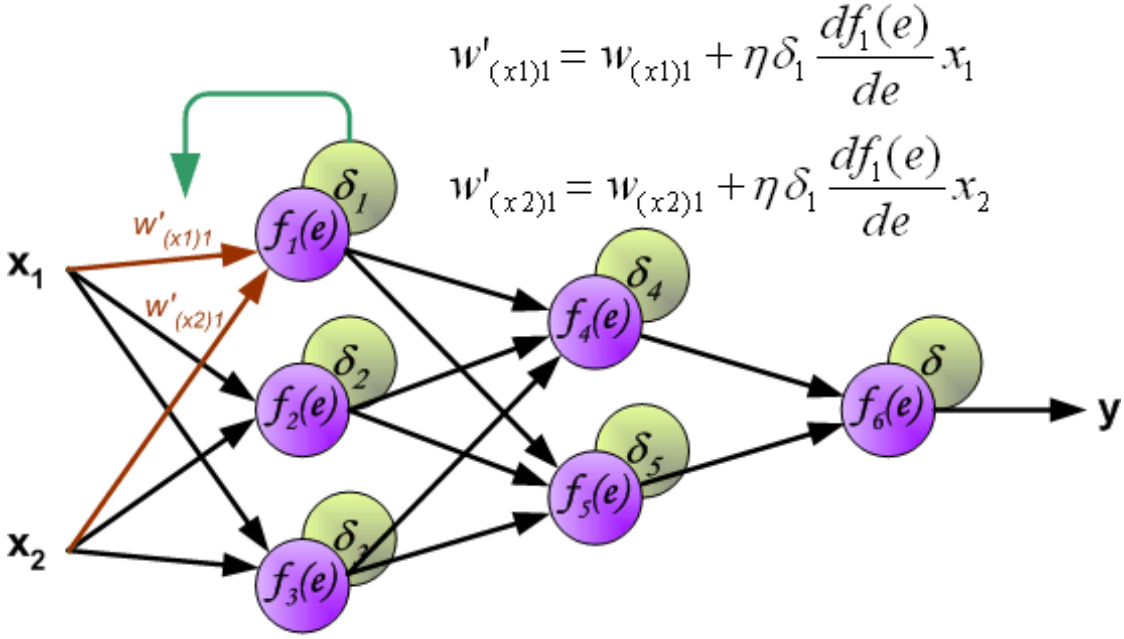

Рисунок 2.10

3. Опис реалізації програмного застосунку

3.1. Обґрунтування вибору засобів розробки

Дивлячись на те, що створювати нейронну мережу ми будемо без використання додаткових модулів, мною була обрана мова розробки Java через її простоту, роботу з класами та наявність стандартних бібліотек для побудови користувацького інтерфейсу.

Для роботи з функціями активації та деактивації було використано стандартний інтерфейс Unaryoperator він дає змогу дуже просто працювати з такими функціями.

Для зчитування файлів було використано клас File, це стандартний клас Java для зчитування та запису файлів.

Для обробки зображення та конвертування їх в масиви було використано клас BufferedImage, який наслідується від класу Image.

Для розробки користувацького інтерфейсу було використано бібліотеку javax.swing та java.awt

3.2. Розробка програмного продукту

3.2.1. Створення моделі

Першим кроком ми створимо клас, який буде відповідати за один шар нейронів та назвемо його Layer. В ньому зберігаються такі дані: розмір шару, список всіх нейронів шару, список всіх нейронів зміщення та список всіх ваг для наступного шару. Нижче приведено конструктор для даного класу.

```
public Layer(int sizeOfLayer, int sizeOfNextLayer) {
    this.layer_length = sizeOfLayer;
    neurons = new double[sizeOfLayer];
    wt = new double[sizeOfLayer][sizeOfNextLayer];
    displacementNeuron = new double[sizeOfLayer];
}
```

Наступний крок це створення нейронної мережі за допомогою раніше створеного класу шару. Нейронна мережа зберігає такі дані: список всіх шарів нашої мережі та значення швидкості навчання, яке ми розглядали в пункті 2.2.4. Нижче приведено конструктор.

```
public NeuralNetwork(double speedLearn, int... sizesOfLayers) {
    this.speedLearn = speedLearn;
    this.allLayers = new Layer[sizesOfLayers.length];
    for (int i = 0; i < sizesOfLayers.length; i++) {
        int next = 0;
        if (i < sizesOfLayers.length - 1) {
            next = sizesOfLayers[i + 1];
        }
        this.allLayers[i] = new Layer(sizesOfLayers[i], next);
        for (int j = 0; j < sizesOfLayers[i]; j++) {
            this.allLayers[i].displacementNeuron[j] = Math.random() * 2.0 - 1.0;
            for (int k = 0; k < next; k++) {
                this.allLayers[i].wt[j][k] = Math.random() * 2.0 - 1.0;
            }
        }
    }
}
```

Значення нейронів зміщення та ваг коливаються в межах проміжку $[-1; 1]$.

Наступним кроком була розроблена функція для заповнення всіх шарів мережі значеннями, а саме значення нейрона дорівнює значення вхідного нейрона з попереднього шару помножено на відповідні вхідні ваги. Потім додається нейрон зміщення та застосовується функція активації(сигмоїда) для того, щоб вихідне значення нейрона було в межах проміжку $[-1; 1]$. Нижче приведена реалізація даної функції.

```
public double[] outputLayer(double[] inputs) {
    UnaryOperator<Double> sigmoidActiovationFunction = x -> 1 / (1 + Math.exp(-x));
    //копіюємо значення пікселів в перший шар нейронів
    System.arraycopy(inputs, srcPos: 0, allLayers[0].neurons, destPos: 0, inputs.length);
    //рахуємо значення для всіх вихідних шарів з урахуванням нейрона зміщення
    for (int i = 1; i < allLayers.length; i++) {
        Layer layer = allLayers[i];
        Layer previousLayer = allLayers[i - 1];
        for (int j = 0; j < layer.layer_length; j++) {
            layer.neurons[j] = 0;
            //новий шар нейронів = множимо значення вихідних нейронів на ваги
            for (int k = 0; k < previousLayer.layer_length; k++) {
                layer.neurons[j] += previousLayer.neurons[k] * previousLayer.wt[k][j];
            }
            //застосовуємо формулу активації та додаємо нейрон зміщення
            layer.neurons[j] += layer.displacementNeuron[j];
            layer.neurons[j] = sigmoidActiovationFunction.apply(layer.neurons[j]);
        }
    }
    return allLayers[allLayers.length - 1].neurons; //повертаємо останній шар вихідних нейронів
}
```

Наступний крок це реалізація методу для алгоритму зворотного розповсюдження помилки. В якому ми вираховуємо величину помилки для кожного нейрона вихідного шару, а потім за допомогою алгоритму, описаного вище, корегуємо значення ваг та нейронів зміщення для кожного шару нашої мережі.

```java
//метод зворотного розповсюдження помилки
public void backpropagationAlgorithm(double[] aims) {
    UnaryOperator<Double> dsigmoidDeactionvationFunction = y -> y * (1 - y);
    double[] allErr = new double[allLayers[allLayers.length - 1].layer_length];
    //рахуємо помилку для кожного нейрону
    for (int index = 0; index < allLayers[allLayers.length - 1].layer_length; index++) {
        //помилка для кожного нейрону = потрібне значення - значення нейрона
        allErr[index] = aims[index] - allLayers[allLayers.length - 1].neurons[index];
    }
    for (int k = allLayers.length - 2; k >= 0; k--) {
        Layer previousLayer = allLayers[k];
        Layer layer = allLayers[k + 1];
        double[] nextErr = new double[previousLayer.layer_length];
        double[] deactivateNeurons = new double[layer.layer_length];
        //коефіцієнт для кожного нейрона в вихідному шарі
        for (int i = 0; i < layer.layer_length; i++) {
            deactivateNeurons[i] = allErr[i] * dsigmoidDeactionvationFunction.apply(allLayers[k + 1].neurons[i]);
            deactivateNeurons[i] *= speedLearn;
        }
        //рахуємо коефіцієнт без врахування вагів для кожного нейрона, який входив в вихідний
        double[][] koef = new double[layer.layer_length][previousLayer.layer_length];
        for (int index = 0; index < layer.layer_length; index++) {
            for (int j = 0; j < previousLayer.layer_length; j++) {
                koef[index][j] = deactivateNeurons[index] * previousLayer.neurons[j];
            }
        }
        //помилка для кожного нейрону не вихідного шару
        for (int index = 0; index < previousLayer.layer_length; index++) {
            nextErr[index] = 0;
            for (int jndex = 0; jndex < layer.layer_length; jndex++) {
                nextErr[index] += previousLayer.wt[index][jndex] * allErr[jndex];
            }
        }
        allErr = new double[previousLayer.layer_length];
        System.arraycopy(nextErr, srcPos: 0, allErr, destPos: 0, previousLayer.layer_length);
        //корекція вагів для кожного вхідного шару
        double[][] weightsNew = new double[previousLayer.wt.length][previousLayer.wt[0].length];
        for (int index = 0; index < layer.layer_length; index++) {
            for (int jndex = 0; jndex < previousLayer.layer_length; jndex++) {
                weightsNew[jndex][index] = previousLayer.wt[jndex][index] + koef[index][jndex];
            }
        }
        previousLayer.wt = weightsNew;
        //корекція нейрону зміщення
        for (int i = 0; i < layer.layer_length; i++) {
            layer.displacementNeuron[i] += deactivateNeurons[i];
        }
    }
}
```

Отже, наша нейронна мережа готова до навчання.

3.2.2. Тренування моделі

Спочатку завантажуємо дані з бази даних рукописних цифр Mnist в якій містяться 60000 картинок різних цифр для тренування.

Потім розбиваємо всі файли на два масиви даних: картинки та відповіді

```
private static void trainFiles(int picturesNumber, BufferedImage[] images, int[] numbers_names) throws IOException {
    File[] files = new File( pathname: "train").listFiles(); //всі картинки
    for (int i = 0; i < picturesNumber; i++) {
        images[i] = ImageIO.read(files[i]); //масив картинок
        numbers_names[i] = Integer.parseInt( s: files[i].getName().charAt(10) + ""); //масив відповідей
    }
}
```

Далі нам потрібно обробити зображення та конвертувати їх у двовимірні масиви пікселів та зберегти в новому масиві даних.

```
    //значення для першого шару
    double[][] inputs = new double[picturesNumber][784];
    //конвертуємо картинки в масиви пікселів
    for (int i = 0; i < picturesNumber; i++) {
        for (int j = 0; j < 28; j++) {
            for (int k = 0; k < 28; k++) {
                inputs[i][j+k*28] = (allImages[i].getRGB(j, k) & 0xff)/255.0;
            }
        }
    }
```

Наступним кроком ми проводимо 3000 епох навчання в яких даємо нейронній мережі на вхід по 100 випадкових картинок з бази в кожній епосі. В кожній ітерації одної картинки кожен її піксель подається на вхід в перший шар, який складається з 784 нейронів по кількості пікселів в картинці. На їх основі ми заповнюємо всю нейронну мережу значеннями, використовуючи раніше написану функцію.

Після цього знаходимо максимальне значення на вихідному шарі, який складається з 10 нейронів, які відповідають конкретній цифрі. Це максимальне значення називається гіпотезою щодо того яка цифра зображена на картинці. Якщо відповідь правильна, ми додаємо кількість правильних відповідей.

Наступним кроком ми використовуємо алгоритм зворотного розповсюдження помилки, який реалізували раніше для корекції значень.

На графіку(Рисунок 3.1) ми можемо побачити як нейромережа навчається, синім позначена кількість правильних відповідей, а червоним величина помилки. Видно, що нейронна мережа дуже швидко навчилася, але вона не впевнена в своїх відповідях.

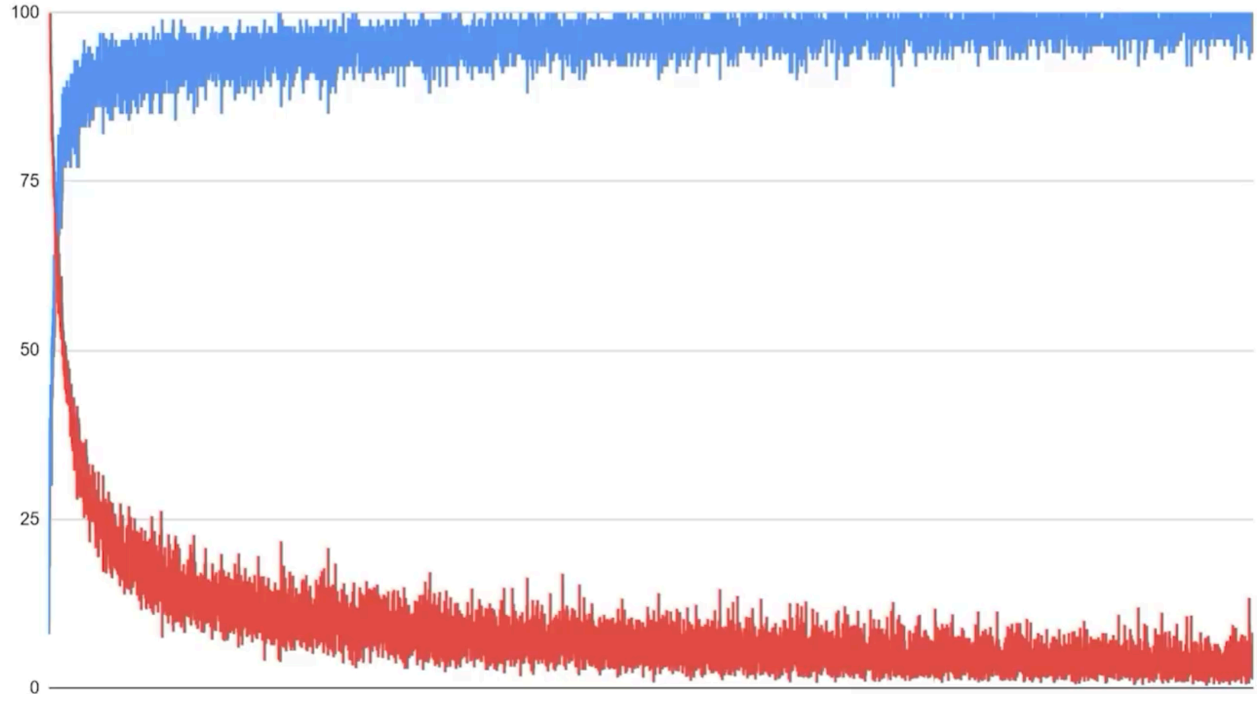

Рисунок 3.1

Після успішного завершення навчання ми записуємо значення нашої навченої нейронної мережі в файл. Потім при кожному новому запуску програми нейронна мережа зчитується з файлу і не навчається кожного разу.

3.3 Тестування програми

При запуску програми відкривається вікно, де є полотно на якому можна малювати. Малювання здійснюється за допомогою лівої кнопки мишки, для того щоб очистити полотно використовується клавіша пробіл. На Рисунок 3.2 можна побачити як це виглядає.

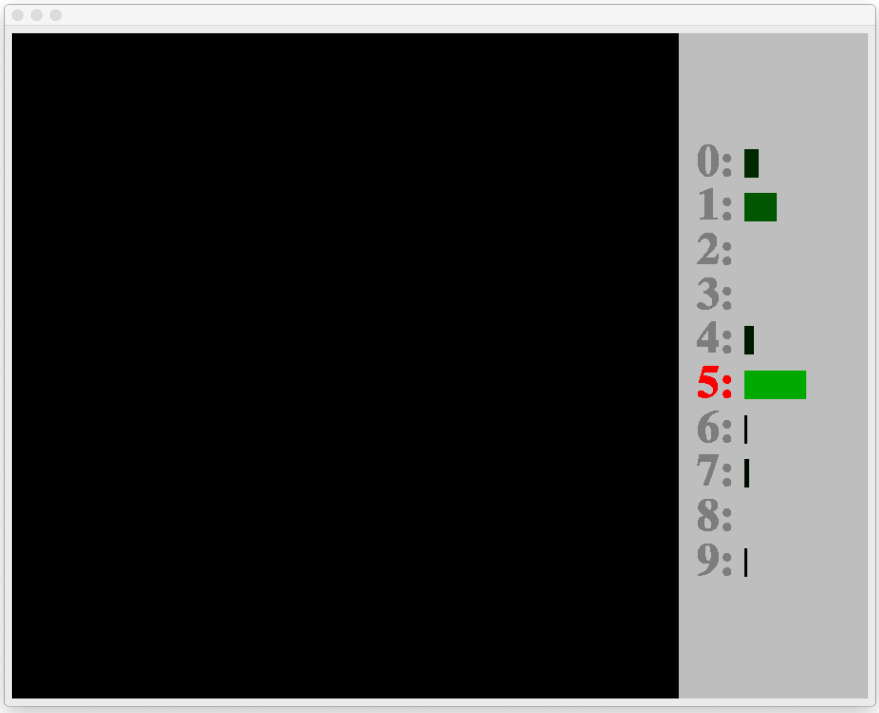

Рисунок 3.2

Через те що розпізнавання проходить в режимі реального часу, нейронна мережа сприймає чорний екран як цифру 5.

На Рисунок 3.3 ми бачимо як програма розпізнає цифри на прикладі двійки в різних варіантах.

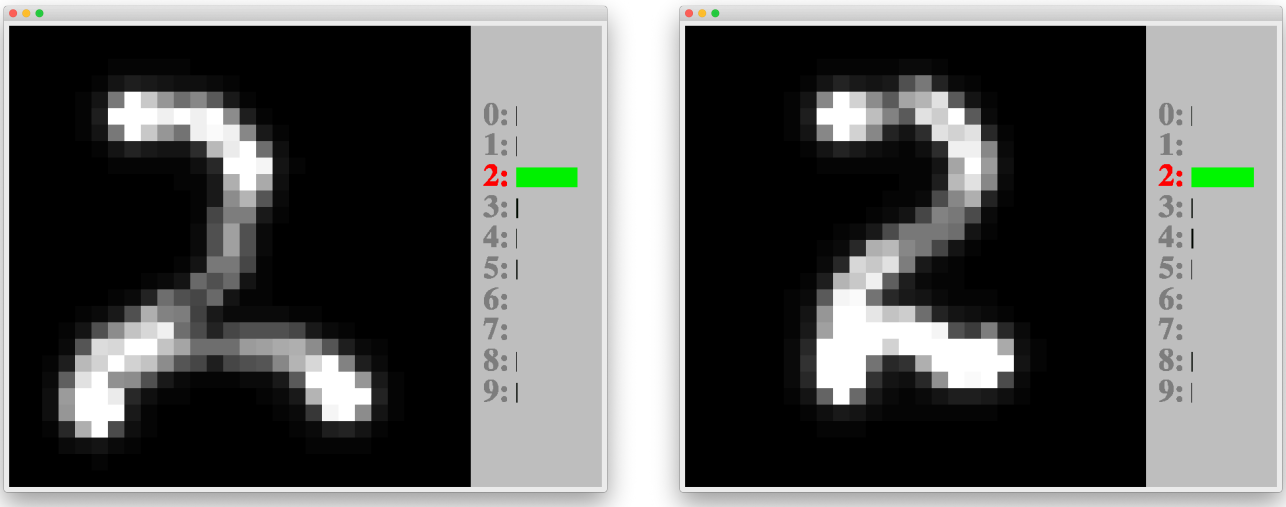

Рисунок 3.3

На Рисунок 3.4 ми можемо бачити випадок, коли програма не на 100% впевнена, що на малюнку зображена цифра 9.

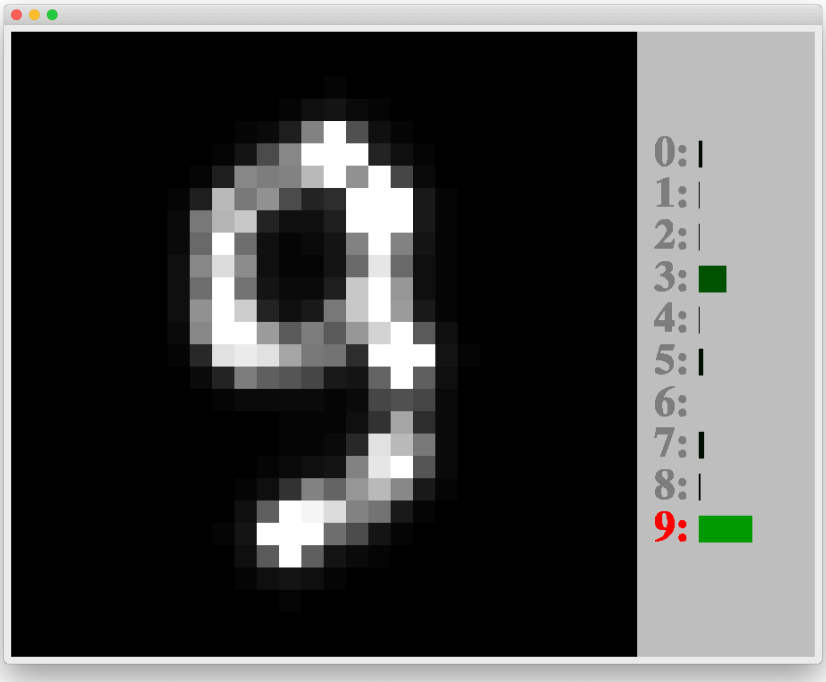

Рисунок 3.4

Не вирішеною проблемою залишаються випадки, коли цифра розташована під кутом(Рисунок 3.5). На даному прикладі цифру 4 розпізнає як 5.

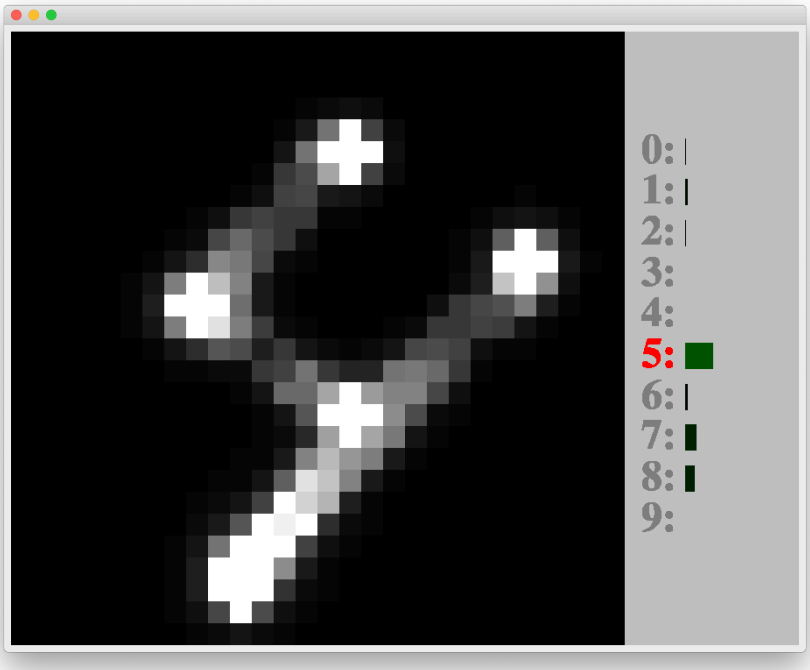

Рисунок 3.5

Висновок

Під час збору інформації та її вивчені було прийнято рішення розробити програму для розпізнавання цифр, в подальшому її можна модифікувати для розпізнавання будь-яких графічних об’єктів.

В результаті роботи мета була досягнута. Було детально розглянуто нейронні мережі, зокрема нейронну мережу на базі архітектури «перцептрон» та реалізована програмно. Також був реалізований користувацький інтерфейс для демонстрації роботи нейронної мережі.

Список джерел

1. ABBYY FineReader [Електронний ресурс]: Матеріал з сайту ABBYY – режим доступу: https://www.abbyy.com/en-eu/finereader/
2. The MNIST database [Електронний ресурс]: режим доступу: http://yann.lecun.com/exdb/mnist/
3. Нейронные сети: теория и практика[Електронний ресурс]: Матеріал з сайту InTalent – режим доступу: https://intalent.pro/article/neyronnye-seti-teoriya-i-praktika.html
4. Машинное обучение[Електронний ресурс]: Матеріал з сайту Machinelearning – режим доступу http://www.machinelearning.ru/wiki/index.php?title=Машинное_обучение
5. Нейрокомпьютерная техника: теория и практика. Автор – Ф. Уоссермен.
6. Principles of training multi-layer neural network using backpropagation[Електронний ресурс] – режим доступу: http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html
7. An overview of gradient descent optimization algorithms [Електронний ресурс]: – режим доступу: https://ruder.io/optimizing-gradient-descent/index.html
8. What is OCR and OCR technology[Електронний ресурс]: Матеріал з сайту ABBYY – режим доступу: https://www.abbyy.com/en-us/finereader/what-is-ocr/
9. Офіційний сайт Java[Електронний ресурс]: https://docs.oracle.com/en/java/
10. Нейронные сети: основные модели. Автор – І. В. Заєнцев.

Додаток А

(створення графічного інтерфейсу)

```
public class UI extends JFrame implements Runnable, MouseListener, MouseMotionListener, KeyListener {

    private int frame = 0;
    private final int widthh = 28;
    private final int heightt = 28;
    private int press = 0;
    private final int increase = 25;
    private int mousex = 0;
    private int mousey = 0;
    private double[][] drawLayout = new double[widthh][heightt];

    private BufferedImage pimage = new BufferedImage(widthh, heightt, BufferedImage.TYPE_INT_RGB);
    private BufferedImage image = new BufferedImage( width: widthh * increase + 200,  height: heightt * increase, BufferedImage.TYPE_INT_RGB);

    private NeuralNetwork neuralNetwork;

    //конструктор
    public UI() throws IOException {
        this.neuralNetwork = readFromFile();
        this.setSize( width: widthh * increase + 200 + 16,  height: heightt * increase + 38);
        this.setVisible(true);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation( x: 50,  y: 50);
        this.add(new JLabel(new ImageIcon(image)));
        addMouseMotionListener( l: this);
        addKeyListener( l: this);
        addMouseListener( l: this);
    }

    @Override
    public void run() {
        while (true) {
            this.repaint();
        }
    }
```

```
    @Override
    public void paint(Graphics g) {
        double[] inputs = new double[784];
        for (int i = 0; i < widthh; i++) {
            for (int j = 0; j < heightt; j++) {
                if (press != 0) {
                    double d = (i - mousex) * (i - mousex) + (j - mousey) * (j - mousey);
                    if (d < 1) {
                        d = 1;
                    }
                    d = d * d;
                    if (press == 1) {
                        drawLayout[i][j] += 0.1 / d;
                    }
                    else {
                        drawLayout[i][j] -= 0.1 / d;
                    }
                    if (drawLayout[i][j] > 1) {
                        drawLayout[i][j] = 1;
                    }
                    if (drawLayout[i][j] < 0) {
                        drawLayout[i][j] = 0;
                    }
                }
                int color = (int) (drawLayout[i][j] * 255);
                color = (color << 16) | (color << 8) | color;
                pimage.setRGB(i, j, color);
                inputs[i + j * widthh] = drawLayout[i][j];
            }
        }
        double[] outputs = neuralNetwork.outputLayer(inputs);
        int maxDigit = 0;
        double maxDigitWeight = -1;
        for (int i = 0; i < 10; i++) {
            if (outputs[i] > maxDigitWeight) {
                maxDigitWeight = outputs[i];
                maxDigit = i;
            }
```

```
        }
    }
    Graphics2D graphics1 = (Graphics2D) image.getGraphics();
    graphics1.drawImage(pimage, x: 0, y: 0, width: widthh * increase, height: heightt * increase, observer: this);
    graphics1.setColor(Color.lightGray);
    graphics1.fillRect( x: widthh * increase + 1, y: 0, width: 200, height: heightt * increase);
    graphics1.setFont(new Font( name: "TimesRoman", Font.BOLD, size: 48));
    for (int i = 0; i < 10; i++) {
        if (maxDigit == i) graphics1.setColor(Color.RED);
        else graphics1.setColor(Color.GRAY);
        graphics1.drawString( str: i + ":", x: widthh * increase + 20, y: i * widthh * increase / 15 + 150);
        Color rectColor = new Color( r: 0, (float) outputs[i], b: 0);
        int rectWidth = (int) (outputs[i] * 100);
        graphics1.setColor(rectColor);
        graphics1.fillRect( x: widthh * increase + 70, y: i * widthh * increase / 15 + 122, rectWidth, height: 30);
    }
    g.drawImage(image, x: 8, y: 30, width: widthh * increase + 200, height: heightt * increase, observer: this);
    frame++;
}

@Override
public void mouseMoved(MouseEvent e) {
    mousex = e.getX() / increase;
    mousey = e.getY() / increase;
}

@Override
public void mousePressed(MouseEvent e) {
    press = 1;
    if (e.getButton() == 3) press = 2;
}

@Override
public void keyPressed(KeyEvent e) {
    if (e.getKeyCode() == KeyEvent.VK_SPACE) {
        drawLayout = new double[widthh][heightt];
    }
}
```