

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіoeлектроніки,
Факультет комп'ютерних наук і технологій
(повне найменування інституту, назва факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему РОЗРОБКА І ДОСЛІДЖЕННЯ СИСТЕМИ РОЗПІЗНАВАННЯ
СИМВОЛІВ СЕНСОРНОГО ВВODУ З ВИКОРИСТАННЯМ
МАШИННОГО НАВЧАННЯ
DEVELOPMENT AND RESEARCH OF A TOUCH INPUT CHARACTER
RECOGNITION SYSTEM USING MACHINE LEARNING

Виконав: студент(ка) 2 курсу, групи КНТ-120м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

Верьовкін Д.С.

(прізвище та ініціали)

Керівник Зеленьова І.Я.

(прізвище та ініціали)

Рецензент Терещенко Е.В.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ІРЕ, ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
“ ” 2021 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Верьовкіна Дмитра Сергійовича
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка і дослідження системи розпізнавання символів сенсорного вводу з використанням машинного навчання.
Development and Research of a Touch Input Character Recognition System Using Machine Learning

керівник проєкту (роботи) Зеленьова Ірина Яківна, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “5” листопада 2021 року №435

2. Строк подання студентом проєкту (роботи) 01 грудня 2021 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Матеріали і методи. 3. Розробка архітектури програми. 4. Основні рішення щодо реалізації компонентів системи. 5. Експлуатація, тестування та експериментальне дослідження.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	Зеленьова І.Я., доцент		
Нормоконтроль	Белова А.В., асистент		

7. Дата видачі завдання “ 6 ” вересня 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2 тиждень	Розділ 1
3	Аналіз методів	3–4 тижні	Розділ 2
4	Розробка архітектури програми	5 тиждень	Розділ 3
5	Розробка програми	6–7 тижні	Розділ 4
6	Дослідження архітектури системи	8 тиждень	Розділ 5
7	Тестування програмного забезпечення	9 тиждень	Розділ 5
8	Оформлення пояснювальної записки	10-12 тиждень	Додатки
9	Нормоконтроль та рецензування	13 тиждень	
10	Захист роботи	14 тиждень	

Студент(ка)

(підпис)

Верьовкін Д.С.

(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис)

Зеленьова І.Я.

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи магістра: 89 с., 46 рис., 5 табл., 3 дод., 17 джерел.

НЕЙРОННА МЕРЕЖА, ПЛАГІН, C++, ІГРОВИЙ РУШІЙ, UNREAL ENGINE 4.

Об'єкт дослідження – програмні засоби для розпізнавання символів.

Предмет дослідження – нейронні мережі для розпізнавання рукописних символів.

Мета роботи – підвищення рівня автоматизації, точності та швидкості навчання процесу розпізнавання рукописних символів шляхом створення програмної системи на основі методів і моделей штучного інтелекту.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування, рушій Unreal Engine 4, мова програмування C++, інтегроване середовище розробки Microsoft Visual Studio 2019, персональний комп'ютер під управлінням операційної системи Microsoft Windows 10.

Результати. Створено програмний застосунок, який у формі плагіна підключається до ігрового рушія Unreal Engine 4 і вирішує задачу створення та навчання нейронної мережі.

Висновки. Розроблено програмну систему для розпізнавання рукописних символів, яка автоматизує та прискорює процес створення та навчання нейронної мережі.

Галузь використання – розробка ігор для комп'ютерних і мобільних платформ з символьним керуванням.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 89 pages, 46 figures, 5 tables, 3 appendixes, 17 sources.

NEURAL NETWORK, PLUGIN, C++, GAME ENGINE, UNREAL ENGINE 4.

The object of research is software for character recognition.

The subject of research is neural networks for handwriting recognition.

The purpose of the work is to increase the level of automation, accuracy and speed of learning the process of handwriting recognition by creating a software system based on methods and models of artificial intelligence.

Materials, methods and technical means: object-oriented programming, Unreal Engine 4 engine, C ++ programming language, Microsoft Visual Studio 2019 integrated development environment, personal computer running Microsoft Windows 10 operating system.

Results. A software application has been created that connects to the Unreal Engine 4 game engine in the form of a plugin and solves the problem of creating and learning a neural network.

Conclusions. A software system for handwritten character recognition has been developed, which automates and accelerates the process of creating and learning a neural network.

Field of application – development of games for computer and mobile platforms with symbolic control.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок.....	8
Вступ.....	9
1 Аналіз предметної області.....	11
1.1 Нейронні мережі.....	11
1.2 Огляд програмних засобів створення нейронних мереж.....	15
1.3 Вибір ігрового рушія.....	18
1.4 Постановка завдання.....	20
1.5 Висновки за розділом 1.....	21
2 Матеріали і методи.....	22
2.1 Функції активації.....	22
2.2 Методи навчання нейронної мережі	25
2.3 Алгоритм побудови багатошарового персептрона.....	32
2.4 Висновки за розділом 2.....	33
3 Розробка архітектури програми.....	34
3.1 Опис основних вимог до програми	34
3.2 Вибір мови програмування	34
3.3 Вимоги до технічних засобів	36
3.4 Проєктування розробки	37
3.5 Висновки за розділом 3.....	41
4 Основні рішення щодо реалізації компонентів системи.....	42
4.1 Програмне забезпечення.....	42
4.2 Висновки за розділом 4.....	47
5 Експлуатація, тестування та експериментальне дослідження.....	48
5.1 Призначення програми	48
5.2 Тестовий набір даних	48
5.3 Результати тестування	48
5.4 Тест кейси для розробленого ПЗ	52
5.5 Опис роботи з програмою	53

5.6 Висновок за розділом 5.....	58
Висновки	59
Перелік джерел посилання.....	60
Додаток А Технічне завдання.....	62
Додаток Б Текст програми	65
Додаток В Слайди презентації.....	81

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

CE	–	перехресна ентропія;
MSE	–	середньоквадратичне відхилення;
ReLU	–	випрямлена пряма функція активації;
UE4	–	Unreal Engine 4;
НМ	–	нейронна мережа;
ПЗ	–	програмне забезпечення.

ВСТУП

В даний час ринок мобільних ігор не стоїть на місці та активно розвивається. Якщо у 2015 році ринок ігор порівну ділили між собою мобільні, комп'ютерні та консольні платформи, то в 2021 році частка мобільних ігор становить 59% [1]. Щомісяця у магазинах додатків App Store та Google Play з'являються десятки тисяч нових ігор [2].

Щоб виділитися посеред інших ігор потрібна новизна, нові ігрові механіки чи нові поєднання ігрових механік. Під новизною розуміється новий ігровий контент, незвичайний дизайн гри чи нестандартні способи керування.

Дану роботу присвячено розробці програмного комплексу, який дозволяє реалізувати спосіб сенсорного керування для мобільних ігор шляхом розпізнавання рукописних символів з використанням машинного навчання та побудованого на архітектурі штучних нейронних мереж.

У роботі піднімаються питання складності навчання нейронної мережі, а також пропонуються варіанти вирішення проблеми низької продуктивності шляхом оптимізації архітектури мережі. У процесі дослідження проведено порівняльний аналіз процесу і результатів навчання нейронної мережі з використанням різних функцій активації нейронів та методів розрахунку витрат.

У першому розділі проводиться аналіз предметної області, розглядаються основні поняття нейронних мереж та їх класифікація. Також проводиться огляд існуючих програмних засобів для реалізації нейронних мереж. Проводиться постановка задачі дослідження.

У другому розділі розглядаються методи навчання нейронних мереж, функції активації нейронів прихованого шару та методи визначення витрат.

У третьому розділі проводиться проектування програмної системи, визначається архітектура нейронної мережі і засоби розробки.

У четвертому розділі описується процес розробки, основні класи програмної системи, їх властивості і методи.

У п'ятому розділі проводиться дослідження розробленої системи шляхом експериментального тестування з використанням різних функцій активації нейронів прихованого шару та методів визначення витрат. Визначається оптимальний варіант для розроблюваної системи. Поетапно описується процес роботи користувача з системою.

У результаті виконання роботи було розроблено плагін для Unreal Engine 4, який дозволяє створювати нейронну мережу з заданими параметрами для розпізнання рукописних символів, що може бути використано для реалізації ігрових механік або управління всередині гри.

Наукова новизна роботи полягає в тому, що в роботі досліджено актуальну проблему підвищення рівня автоматизації розпізнання рукописних символів – на основі методів і моделей штучного інтелекту, з метою поліпшення точності та швидкості навчання нейронної мережі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Нейронні мережі

Штучна нейронна мережа (НМ) – спрощена модель біологічної нейронної мережі, що є сукупністю штучних нейронів, які взаємодіють між собою [3].

1.1.1 Штучний нейрон

Штучний нейрон (формальний нейрон, нейроподібний елемент) – це примітивний обчислювальний пристрій (або його модель), що має кілька входів і один вихід, і є основним обчислювальним елементом НМ [3]-[5]. Схему штучного нейрона зображено на рис. 1.1.

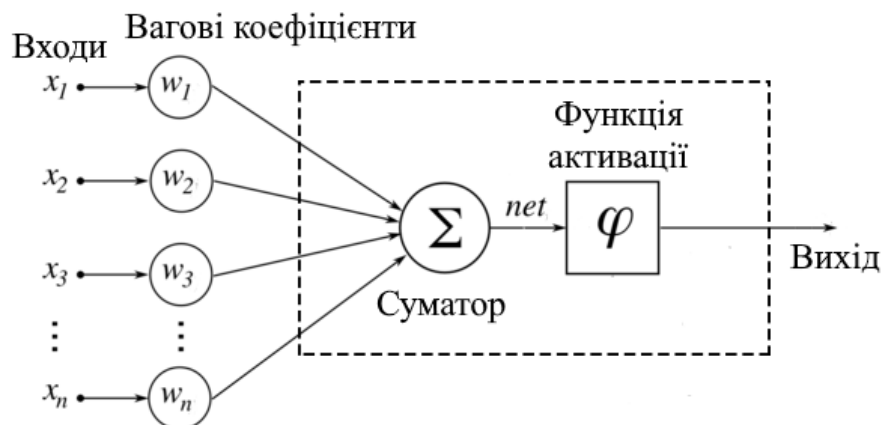


Рисунок 1.1 – Структура штучного нейрона [3]

У нейрона є n входів x_i , у кожного з якого є вага w_i , на який множиться сигнал, що проходить у зв'язку. Після цього зважені сигнали $x_i \cdot w_i$ надсилаються до суматора, який агрегує всі сигнали у зважену суму [3]. Цю суму також називають net і представлена на формулі (1.1).

$$net = \sum_{i=1}^N w_i \cdot x_i = w^T \cdot x, \quad (1.1)$$

де x_i – значення входу;

w_i – вага входу;

w^T – транспонована матриця ваг.

Нейрон повинен якось обробити виважену суму net та сформувати адекватний вихідний сигнал. Для цих цілей використовують функцію активації, яка перетворює зважену суму в якесь число, яке буде виходом нейрона. Функція активації позначається $\varphi(net)$ [3]. Таким чином, формальний нейрон реалізує скалярну функцію векторного аргументу, моделюючи перетворення вхідних сигналів у синапсах та тілі біологічного нейрона [4].

Набір вагових коефіцієнтів нейрона w моделює його пам'ять. Тому нейрони можна розглядати як запам'ятовуючі пристрої. У той же час нейрони можуть розглядатися як примітивні процесори, що здійснюють обчислення значення функції активації на основі значення дискримінантної функції вхідних сигналів і ваг [5].

1.1.2 Одношаровий перцептрон

Один нейрон може виконувати найпростіші обчислення, але основні функції нейромережі забезпечуються окремими нейронами, а сполуками між ними. Одношаровий перцептрон є найпростішою мережею, що складається з групи нейронів, що утворюють шар, зображений на рис. 1.2.

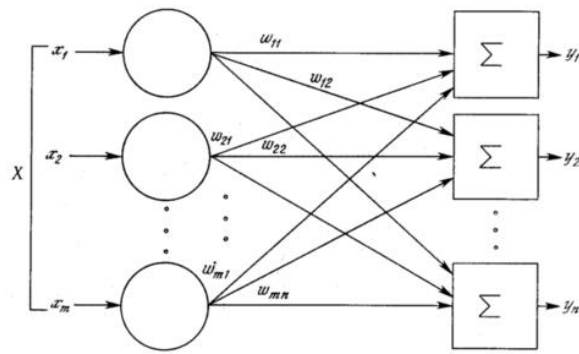


Рисунок 1.2 – Одношаровий перцепрон [6]

Вхідні дані кодуються вектором значень, кожен елемент подається відповідний вхід кожного нейрона в шарі. У свою чергу нейрони обчислюють вихід незалежно один від одного. Розмірність виходу (тобто кількість елементів) дорівнює кількості нейронів, а кількість синапсів у всіх нейронів має бути однаковим і збігатися з розмірністю вхідного сигналу [5]-[7].

1.1.3 Багатошаровий персептрон

Багатошарова нейронна мережа (персептрон) – це нейронна мережа, що складається з вхідного, вихідного та розташованих між ними одного (або декількох) прихованих шарів нейронів [3]. Багатошарова нейронна мережа з одним прихованим шаром зображена на рис 1.3.

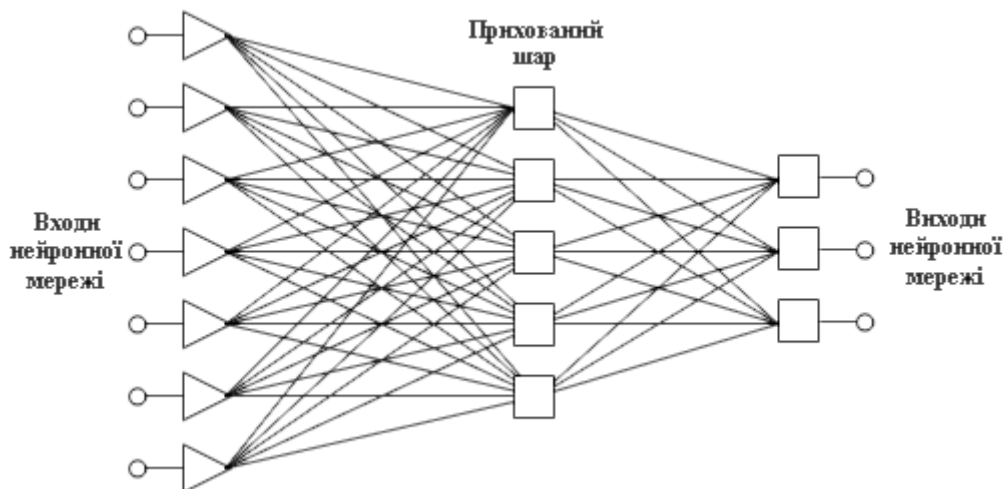


Рисунок 1.3 – Багатошарова нейронна мережа [7]

Мережі прямого поширення – штучні нейронні мережі, в яких сигнал поширюється суворо від вхідного шару до вихідного. У зворотному напрямку сигнал не поширюється.

Усі мережі, описані вище, були мережами прямого поширення, як впливає з визначення. Такі мережі широко використовуються і цілком успішно вирішують певний клас завдань: прогнозування, кластеризація та розпізнавання. Однак сигнал у нейронних мережах може йти і у зворотний бік [4].

Мережі із зворотними зв'язками – штучні нейронні мережі, у яких вихід нейрона може знову подаватися з його вхід. У загальному випадку це означає можливість поширення сигналу від виходів до входів [5]. Приклад такої мережі зображено на рис. 1.4.

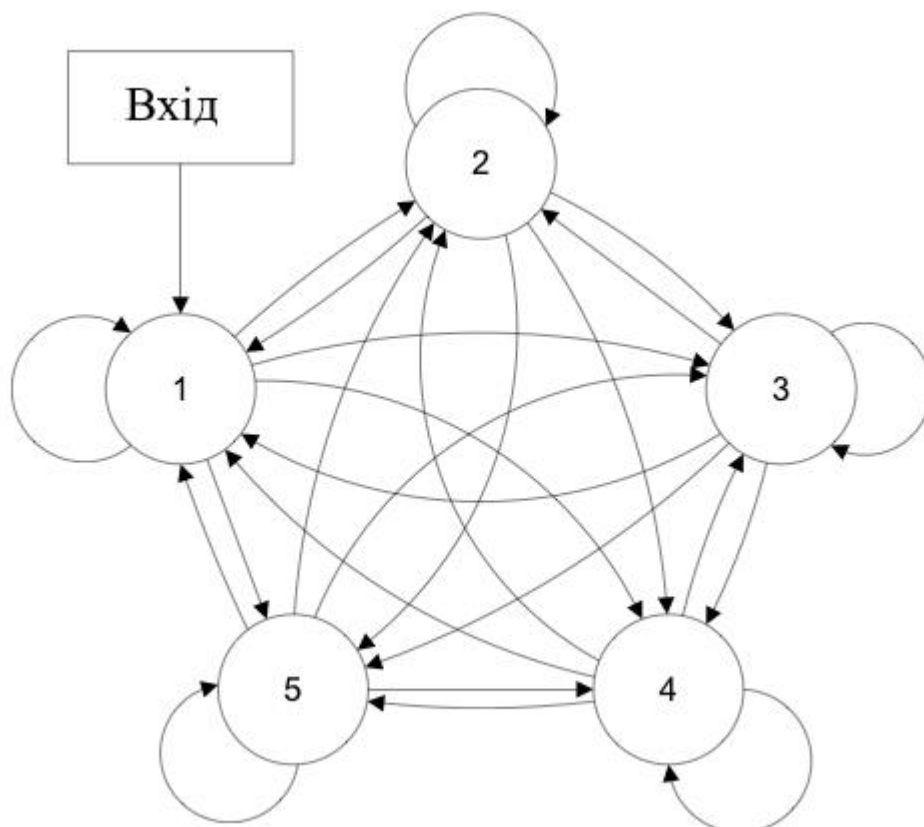


Рисунок 1.4 – НМ зі зворотними зв'язками [5]

У мережах прямого поширення вихід мережі визначається вхідним сигналом та ваговими коефіцієнтами при штучних нейронах. У мережах із зворотними зв'язками виходи нейронів можуть повертатися на входи. Це означає, що вихід якогось нейрона визначається не тільки його вагами та вхідним сигналом, але ще й попередніми виходами (оскільки вони знову повернулися на входи) [3].

Нейронні мережі використовуються для вирішення складних завдань, які вимагають аналітичних обчислень. Найбільш поширені види використання нейронних мереж:

- класифікація – розподіл даних за параметрами;
- передбачення – здатність передбачити наступний крок;
- розпізнавання. В даний час є найширшим застосуванням нейронних мереж [5].

1.2 Огляд програмних засобів створення нейронних мереж

Кількість доступних бібліотек для глибокого навчання постійно зростає.

У 2018 році було проведено порівняння популярних фреймворків з відкритим вихідним кодом для розробки нейронних мереж [6]. Оцінювалися наступні критерії:

- онлайн списки вакансій;
- огляд використання KDnuggets;
- Google Search Volume;
- середні статті;
- книги Amazon;
- статті ArXiv;
- активність GitHub.

Результати порівняння та фінальний бал показано на рис 1.5.

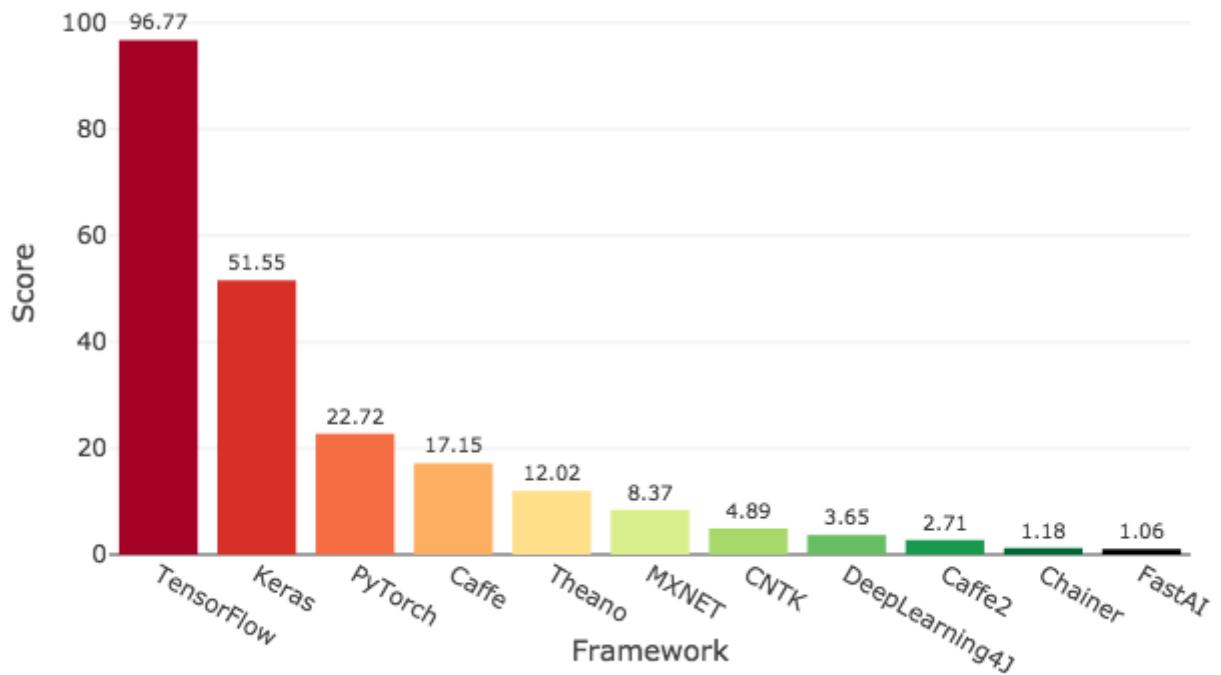


Рисунок 1.5 – Загальна оцінка фреймворків за 2018 рік [6]

За результатами видно, що TensorFlow значно опереджає своїх конкурентів за темпом зростання та кількістю навчального матеріалу.

В даний час найвідомішою бібліотекою глибокого навчання у світі є TensorFlow. Продукт Google використовує машинне навчання у всіх своїх продуктах для покращення пошукової системи, перекладу, підпису до зображень або рекомендацій.

TensorFlow – це бібліотека з відкритим вихідним кодом для цифрових обчислень, розроблена командою Google Brain Team у дослідницькій організації Google's Machine Intelligence [7].

Її центральна ідея базується на графах потоків даних. Вузли в графі є математичними операціями, а ребра графа – багатовимірні масиви даних (тензори), поширені між ними.

TensorFlow можна використовувати у програмах, написаних на C#, C++, Python, Java та JavaScript.

Для тренування нейронної мережі можна скористатися власним персональним комп'ютером чи хмарним рішенням від Google. Для навчання моделі можна використовувати як графічний, так і центральний процесори [8].

Запускати навчену модель можна на комп'ютерах під керуванням Windows, MacOS або Linux, хмарних застосунках, мобільних пристроях iOS та Android [7], [9].

TensorFlow також надає графічну візуалізацію графа (інструмент TensorBoard) та доступ до рівнів обчислень для налагодження за потреби.

Порівняльна таблиця з двома іншими популярними фреймворками представлена у табл. 1.1.

Таблиця 1.1 – Порівняння фреймворків для нейронних мереж

	TensorFlow	Keras	PyTorch
Тип графу	Статичний	Статичний/динамічний	Динамічний
Рівень API	Низький і високий	Високий	Високий
Мови програмування	Python, C/C++, Java, Go, JavaScript, R, Julia, Swift	Python, R	Python, C++, Julia
Підтримка власних операцій	C++	Python	Python, C++
Швидкість навчання на GPU	Середня	Низька	Висока
Швидкість навчання на CPU	Висока	Низька	Середня

Як видно з таблиці порівняння, TensorFlow є найбільш вдалим вибором для розробки нейронної мережі, завдяки великій кількості підтримуваних мов програмування, швидкому навчанні на центральному процесорі, активній

підтримці розробки фреймворку, великій кількості навчальних матеріалів і якісній документації.

1.3 Вибір ігрового рушія

Основними засобами розробки ігор є ігрові рушії, які відповідають за низькорівневий опис фізики гри, правил рендерингу графіки та загальну взаємодію між елементами гри [10].

В даний час існує безліч різних ігрових рушіїв. Основні їх відмінності полягають у мовах програмування, функціональності і, що не менш важливо, у вартості ліцензії.

Unreal Engine 4 - ігровий рушій, що розробляється і підтримується компанією Epic Games. Перша гра, створена на цьому рушійі з'явилася в 1998 році. З тих пір різні версії двигуна були використані в більш ніж сотні ігор та інших проєктів. Написаний мовою C++, двигун дозволяє створювати ігри більшості операційних систем і платформ: Microsoft Windows, Linux, Mac OS і Mac OS X; консолей Xbox, Xbox 360, Xbox One, PlayStation 2, PlayStation 3, PlayStation 4, PSP, PS Vita, Wii, Dreamcast, GameCube та ін., а також на пристроях керованих системою iOS та Android [10], [11].

Для спрощення портування двигун використовує модульну систему залежних компонентів: підтримує різні системи рендерингу (Direct3D, OpenGL, Pixomatic), відтворення звуку (EAX, OpenAL, DirectSound3D), засоби голосового відтворення тексту, розпізнавання мовлення, модулі для роботи з мережею та підтримки різних пристроїв введення [11].

Основною мовою програмування є C++, але для швидкого прототипування або для налаштування ассетів використовуються скриптова мова візуального програмування Blueprints.

Є великий магазин ассетів як з платним контентом, так і з безкоштовним. Велика бібліотека плагінів дозволяє швидко підключати до проєкту необхідний функціонал.

Вбудований плагін Gameplay Ability System дозволяє реалізовувати логіку ігор з класовою механікою та здібностями.

2 березня 2015 року Unreal Engine 4 став безкоштовним. Проте, розробники ігор, як і раніше, повинні передавати 5% від прибутку гри компанії Epic Games, але за умови, що доходи гри становлять понад \$3000 за квартал [11]. Порівняльна характеристика ігрових рушіїв наведена у табл. 1.2.

Таблиця 1.2 – Порівняння ігрових рушіїв

Критерії порівняння	Ігрові рушії	
	Unreal Engine 4	Unity
Мова програмування	C++	C#
Мова візуального програмування	Blueprints	Bolt
Власний магазин ассетів	+	+
Кількість навчальних матеріалів від користувачів	Середня	Велика
Кількість навчальних матеріалів розробників	Велика	Середня
Цінова політика	5% від прибутку, якщо він перевищує 3000\$ у квартал	1800\$/рік за особу, якщо прибуток перевищує 100 000\$/рік
Доступ до вихідного коду	Відкритий	Тільки по передплаті

Набір функцій UE4 краще підходить для створення масштабних 3D проєктів. У нього є вбудовані засоби для автоматичної оптимізації сцени, а також відкритий вихідний код, що дозволяє змінювати рушій під кожний окремий проєкт. Також у UE4 є значна перевага у розробці проєктів з класовою системою. Вона полягає у використанні плагіну Gameplay Ability System, який значно спрощує розробку завдяки спрощеному процесу реалізації навичок та їх взаємодії.

У UE4 є плагін для роботи з TensorFlow, аля для роботи плагіну необхідно встановити ще один плагін для виконання Python коду [13]. Архітектуру підключення плагінів представлена на рис. 1.6.

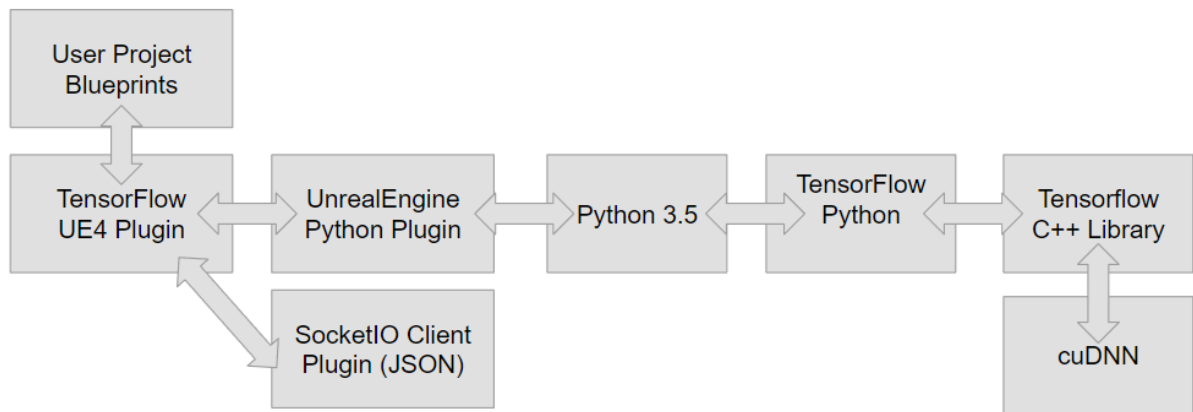


Рисунок 1.6 – Архітектура плагінів [10]

Розробляти програмну систему у вигляді плагіну, яка використовує для своєї роботи два інших плагіни не є розумною ідеєю, так як це створює залежність від інших плагінів, що може затримати перехід на нову версію рушія або навіть зробити її неможливою.

1.4 Постановка завдання

Дослідити та реалізувати НМ на основі багат шарового перцептрону для автоматизації та спрощення процесу навчання для розпізнавання символів. Додатково розглянути функції активації нейронів прихованого шару та їх вплив на поведінку НМ. Дослідити вплив методу розрахунку витрат на швидкість навчання та точність розпізнання НМ.

1.5 Висновки за розділом 1

В результаті аналізу предметної області виявлено, що для задачі розпізнавання символів може бути вирішена за нейронної мережі на основі багатошарового персептрону.

Розглянуті основні ігрові рушії для розробки ігор. Був обраний ігровий рушій UE4. ПЗ буде розроблено у виді плагіну до цього ігрового рушія. В результаті огляду існуючих фреймворків було виявлено, що їх використання не є доцільним для створення програмного застосунку тому, що при використанні фреймворку TensorFlow виникає залежність від двох інших плагінів, необхідних для його функціонування.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Функції активації

Активаційна функція φ має бути монотонною. Зазвичай вона належить інтервалу $[0,1]$ або $[-1,1]$. Розглянемо найчастіше використовувані функції активації [3]-[5].

Порогова функція. Це проста лінійна функція. Якщо вхідне значення менше порогового, то значення функції активації дорівнює мінімальному допустимому, інакше максимально допустимому. Вона приймає значення в діапазоні $[0;1]$ [3]-[5]. Описується у формулі (2.1), та представлена на рис. 2.1.

$$\varphi(net) = \begin{cases} 1, net \geq \theta \\ 0, net < \theta \end{cases} \quad (2.1)$$

де θ – межа активації.

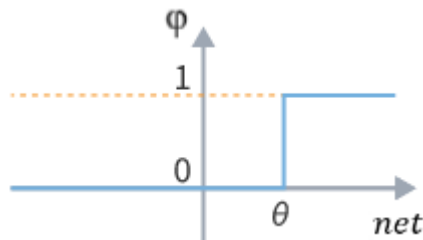


Рисунок 2.1 – Лінійна функція активації [5]

Недолік цієї функції – вона робить область значень виходу нейрона обмеженою і, по суті, зводить усі можливості такої нейронної мережі до вирішення задачі бінарної класифікації. Щоб апроксимувати складніші залежності, потрібно збільшувати кількість бінарних нейронів [5].

Цю проблему частково вирішує застосування функції з лінійним порогом. Вона описується формулою (2.2) та представлена на рис. 2.2:

$$\varphi(net) = \begin{cases} 1, net \geq \theta \\ a \cdot net, 0 \leq net \leq \theta \\ 0, net < 0 \end{cases}, \quad (2.2)$$

де a – крутість зростання.

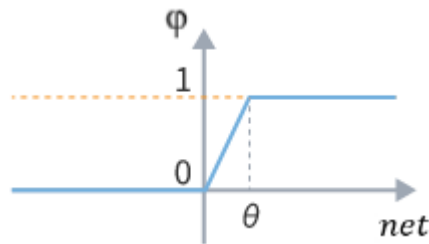


Рисунок 2.2 – Лінійний поріг [5]

Використання активаційної функції у вигляді лінійного порога розширює область значень виходу нейрона, але при цьому він все ще залишається лінійним перетворювачем, що значно знижує апроксимуючі можливості мережі. Крім цього, наявність двох точок розриву, де функція не диференційована, унеможлиблює використання лінійного порога в градієнтних алгоритмах навчання, де використовується похідна активаційної функції [3]-[5].

Тому при навчанні багатошарових нейронних мереж найчастіше використовуються сигмоїдну активаційну функцію [5]. Вона приймає значення $[-1;1]$, представлена на рисунку 2.3 та описується наступною формулою (2.3).

$$\varphi(net) = \frac{1}{1 + e^{-a \cdot net}}, \quad (2.3)$$

де a – крутість зростання.

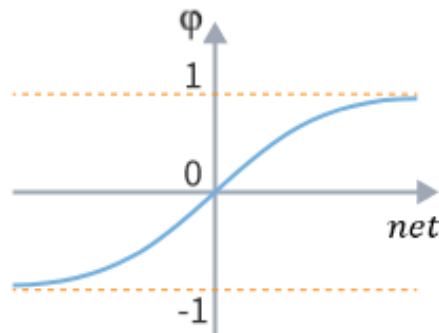


Рисунок 2.3 – логістична функція активації [5]

Різновидом сигмоїдної функції є її скорегована версія гіперболічний тангенс. Він приймає значення $[0;1]$, представлений на рисунку 2.4 та задається формулою (2.4).

$$\varphi(net) = \frac{2}{1 + e^{-2 \cdot a \cdot net}} - 1, \quad (2.4)$$

де a – крутість зростання.

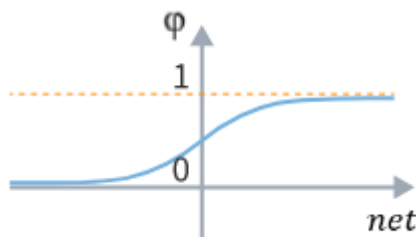


Рисунок 2.4 – Гіперболічний тангенс [5]

Це монотонно зростаючі функції, що диференціюються по всій області визначення, що робить їх застосовними в алгоритмах навчання, що використовують похідні активаційної функції. Зазвичай усі нейрони мережі мають однакову активаційну функцію [5].

Випрямлена лінійна функція активації (ReLU). Функція ReLU є лінійною функцією і зараз вважається набагато більш простим та ефективним варіантом передавальної функції з погляду складності обчислень. Це один із

останніх успіхів у галузі методів оптимізації глибоких нейронних мереж [6].
Графік функції активації наведено на рис. 2.5 і описується формулою (2.5).

$$\text{ReLU}(net) = \begin{cases} net, & net \geq 0 \\ 0, & net < 0 \end{cases}, \quad (2.5)$$

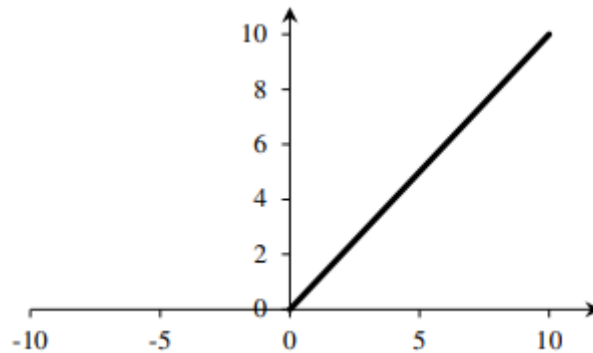


Рисунок 2.5 – Графік функції активації [6]

Її похідна дорівнює одиниці чи нулю, і тому градієнтне зростання чи ослаблення не відбуваються. Крім того, використання цієї функції призведе до зменшення масштабу.

2.2 Методи навчання нейронної мережі

Навчання нейронної мережі – це процес, у якому параметри нейронної мережі налаштовуються за допомогою моделювання середовища, в яке ця мережа вбудована. Тип навчання визначається способом підстроювання параметрів. Розрізняють алгоритми навчання з учителем та без вчителя. Процес навчання з учителем є пред'явленням мережі вибірки навчальних прикладів. Кожен зразок подається на входи мережі, потім проходить обробку всередині структури НМ, обчислюється вихідний сигнал мережі, який порівнюється з відповідним значенням цільового вектора, що являє собою необхідний вихід мережі [3]-[5].

Для того, щоб нейронна мережа була здатна виконати поставлене завдання, її потрібно навчити. Розрізняють алгоритми навчання з учителем та без вчителя. Процес навчання з учителем є пред'явленням мережі вибірки навчальних прикладів. Кожен зразок подається на входи мережі, потім проходить обробку всередині структури НМ, обчислюється вихідний сигнал мережі, який порівнюється з відповідним значенням цільового вектора, що являє собою необхідний вихід мережі. Потім за певним правилом обчислюється помилка, і відбувається зміна вагових коефіцієнтів зв'язків усередині мережі, залежно від обраного алгоритму. Вектори навчальної множини пред'являються послідовно, обчислюються помилки і ваги підлаштовуються для кожного вектора до тих пір, поки помилка по всьому навчальному масиву не досягне допустимо низького рівня [3]-[5]. Процес навчання нейронної мережі представлено на рис. 2.6.

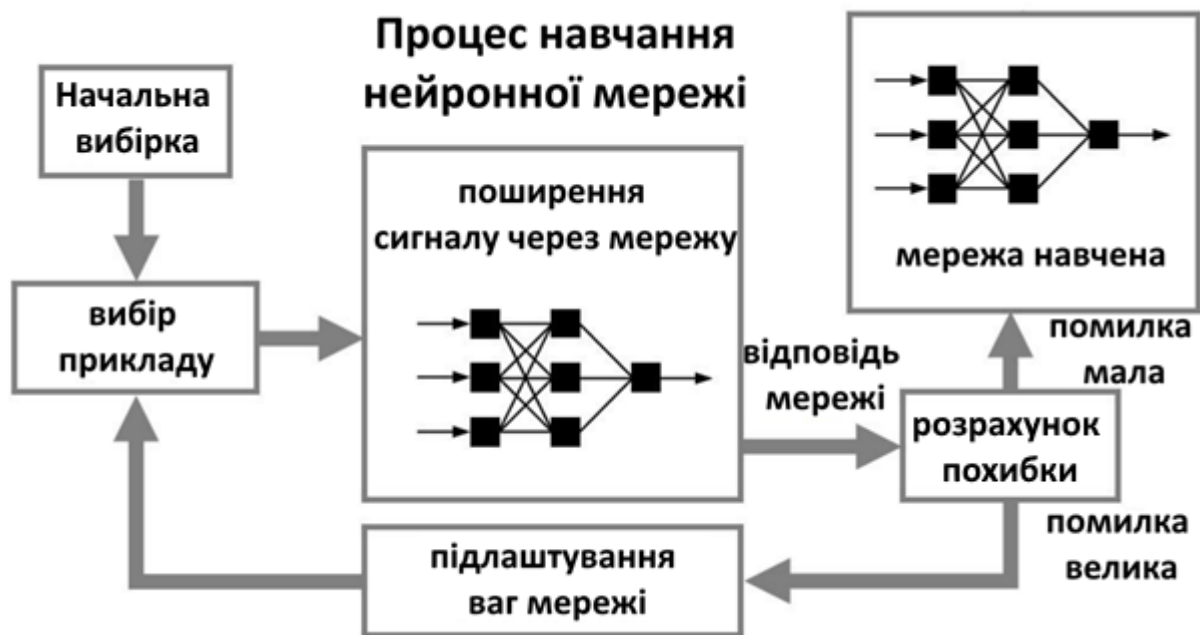


Рисунок 2.6 – Процес навчання нейронної мережі [3]

При навчанні без вчителя навчальна множина складається лише з вхідних векторів. Навчальний алгоритм підлаштовує ваги мережі те щоб узгоджувалися вихідні вектори, тобто. щоб пред'явлення досить близьких

вхідних векторів давало однакові виходи. Процес навчання, отже, виділяє статистичні властивості навчальної множини та групує подібні вектори до класів. Пред'явлення на вхід вектора з даного класу дасть певний вихідний вектор, але до навчання неможливо передбачити, який буде вироблятися даним класом вхідних векторів. Отже, виходи такої мережі мають трансформуватися на деяку зрозумілу форму, зумовлену процесом навчання [3], [4].

Математично процес навчання можна описати в такий спосіб. У процесі функціонування нейронна мережа формує вихідний сигнал Y , реалізуючи певну функцію $Y = G(X)$. Якщо архітектура мережі задана, вид функції G визначається значеннями синаптичних ваг і зміщеної мережі [3].

Нехай рішенням деякої задачі є функція $Y = F(X)$, задана параметрами вхідних-вихідних даних $(X_1, Y_1), (X_2, Y_2), \dots, (X_N, Y_N)$, для яких $Y_k = F(X_k)$ ($k = 1, 2, \dots, N$). Навчання полягає у пошуку функції G , близької до F у сенсі деякої функції помилки E . Якщо вибрано безліч навчальних прикладів – пара (X_N, Y_N) (де $k = 1, 2, \dots, N$) та спосіб обчислення функції помилки E , то навчання нейронної мережі перетворюється на вирішення задачі багатовимірної оптимізації, що має дуже велику розмірність [4].

Для розв'язання цієї задачі можуть використовуватись такі (ітераційні) алгоритми:

а) алгоритми локальної оптимізації з обчисленням приватних похідних першого порядку:

- 1) градієнтний алгоритм (метод якнайшвидшого спуску),
- 2) методи з одновимірною та двовимірною оптимізацією цільової функції у напрямку антиградієнта;
- 3) метод сполучених градієнтів;
- 4) методи, що враховують напрямок антиградієнта на кількох кроках алгоритму;

б) алгоритми локальної оптимізації з обчисленням приватних похідних першого та другого порядку:

- 1) метод Ньютона;
 - 2) методи оптимізації з розрідженими матрицями Гессе;
 - 3) квазиньютонівські методи;
 - 4) метод Гауса-Ньютона;
 - 5) метод Левенберга-Марквардта та ін;
- в) стохастичні алгоритми оптимізації:
- 1) пошук у випадковому напрямку;
 - 2) імітація відпалу;
 - 3) метод Монте-Карло (чисельний метод статистичних випробувань);
- г) алгоритми глобальної оптимізації (завдання глобальної оптимізації вирішуються з допомогою перебору значень змінних, яких залежить цільова функція) [4].

2.2.1 Метод градієнтного спуску

Градієнтний спуск – це метод знаходження локального екстремуму (мінімального чи максимального) функції шляхом переміщення градієнтом. Градієнт являє собою вектор, який визначає крутість схилу функції та вказує його напрямок щодо точки на поверхні або графіці [11]. Щоб знайти градієнт потрібно взяти похідну від функції у цій точці – $f'(w_1)$ і представлена на рис. 2.7.

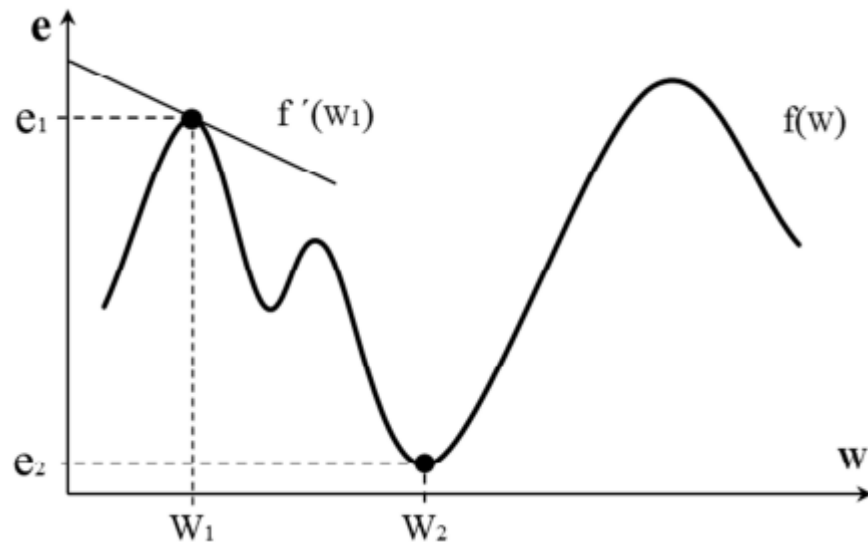


Рисунок 2.7 – Похідна функції [11]

Метод градієнтного спуску у чистому вигляді може «застрягати» у локальних мінімумах функції. Для подолання проблеми методу градієнтного спуску використовується метод моментів. Ідея методу збереження частини швидкості спуску в кожній ітерації. Фактично, відбувається додавання до збільшення ваги збільшення ваги з попередньої ітерації, помноженої на коефіцієнт моменту, що продемонстровано у формулі (2.6) [12].

$$\Delta W_t = \eta \cdot \nabla E + \mu \cdot \Delta W_{t-1}, \quad (2.6)$$

де η – коефіцієнт швидкості навчання;

∇E – градієнт функції втрати;

μ – коефіцієнт моменту;

$\Delta W_t - 1$ – зміна ваги на попередній ітерації.

На додаток моменту метод зворотного поширення також використовує такий параметр, як швидкість навчання. При цьому можна контролювати розмір ваг, що коригують, на кожній ітерації [12].

2.2.2 Алгоритм зворотного розповсюдження помилки

Алгоритм зворотного поширення помилки одна із методів навчання багат шарових нейронних мереж прямого поширення. Навчання алгоритмом зворотного поширення помилки передбачає два проходи по всіх верствах мережі: прямого та зворотного. При прямому проході вхідний вектор подається на вхідний шар нейронної мережі, після чого поширюється мережею від шару до шару. В результаті генерується набір вихідних сигналів, який є фактичною реакцією мережі на даний вхідний образ. Під час прямого проходу всі синаптичні маси мережі фіксовані. Під час зворотного проходу всі синаптичні ваги налаштовуються відповідно до правила корекції помилок, а саме: фактичний вихід мережі віднімається від бажаного, у результаті формується сигнал помилки. Цей сигнал згодом поширюється по мережі у напрямку, зворотному напрямку синаптичних зв'язків [13]. Алгоритм зображено на рисунку 2.8.

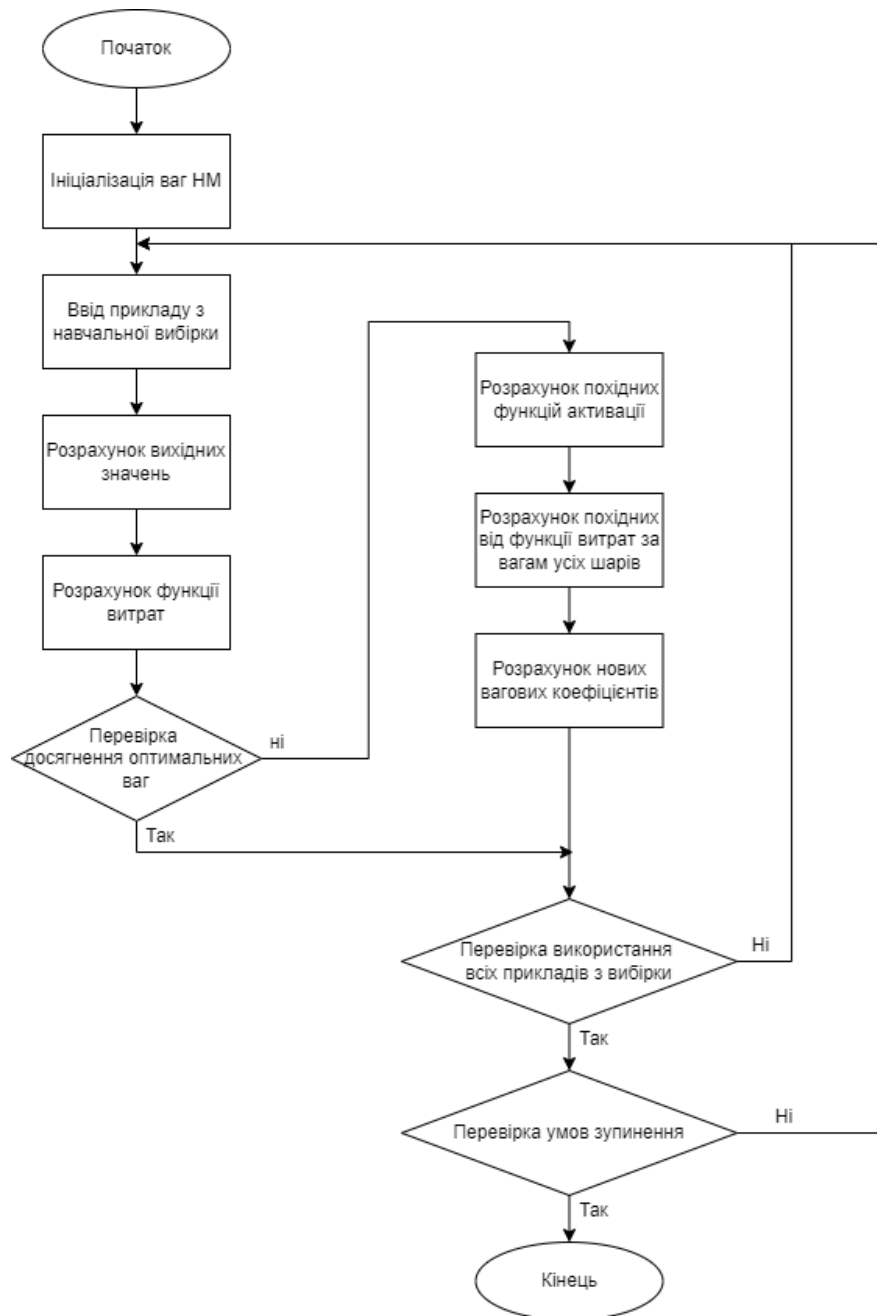


Рисунок 2.8 – Алгоритм зворотного розповсюдження похибки [13]

2.2.3 Цільова функція

Ефективний спосіб кількісної оцінки наближення прогнозування до відповіді дуже важливий під час навчання нейронних мереж. Цільова функція, що диференціюється (також відома як функція витрат) необхідна для виконання зворотного поширення помилки і оновлення всіх відповідних ваг, що впливають на прогнози вихідних даних. У якості функції витрат

використовується функція середньоквадратичного відхилення (MSE), що представлена на формулі (2.7), і функція перехресної ентропії (CE) – формула (2.8) [14].

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2, \quad (2.7)$$

де y_i – фактичне значення;

$\hat{y}_i$ – очікуване значення.

$$CE = - \sum_{i=1}^C y_i \log \hat{y}_i, \quad (2.8)$$

де y_i – фактичне значення;

$\hat{y}_i$ – очікуване значення.

2.3 Алгоритм побудови багат шарового персептрона

Щоб побудувати багат шаровий персептрон, необхідно визначити, який сенс вкладається в компоненти вхідного вектора X . Вхідний вектор повинен містити формалізовану умову завдання, тобто всю інформацію, необхідну для отримання відповіді [3]-[5].

Вибрати вихідний вектор Y таким чином, щоб його компоненти мали повну відповідь для поставленого завдання. Вибрати тип функції активації нейронів. У цьому бажано врахувати специфіку завдання, оскільки успішний вибір збільшить швидкість навчання. Вибрати кількість шарів та нейронів у шарі.

Задати діапазон зміни входів, виходів, ваги та порогових рівнів на основі обраної функції активації. Призначити початкові значення ваги та пороговим рівням. Початкові значення не повинні бути більшими, щоб нейрони не

опинилися в насиченні (на горизонтальній ділянці функції активації), інакше навчання буде дуже повільним. Початкові значення не повинні бути занадто малими, щоб виходи більшої частини нейронів не були рівні нулю, інакше навчання теж сповільниться [3]-[5]..

Провести навчання, тобто підібрати параметри мережі так, щоб завдання вирішувалося якнайкраще. Після закінчення навчання мережа зможе вирішувати завдання того типу, яким вона навчена.

2.4 Висновки за розділом 2

Було проаналізовано методи, які можуть використовуватися для навчання нейронною мережі з архітектурою багатошарового персептрону, а саме метод мінімізації помилки за допомогою градієнтного спуску та метод зворотного розповсюдження помилки. Був сформований алгоритм навчання нейронної мережі, на основі якого буде виконуватися проєктування програмного застосунку.

3 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

3.1 Опис основних вимог до програми

За результатами виконаного огляду було сформульовано наступні функціональні вимоги:

- створення і редагування даних профілів з символами;
- створення і редагування даних символів всередині профілю;
- налаштування параметрів чутливості нейронної мережі;
- відображення процесу навчання нейронної мережі;
- відображення результатів тестування нейронної мережі;
- використання результатів роботи нейронної мережі під час гри (виклик ігрової логіки).

Програмне забезпечення має створювати нейронну мережу, яка коректно розпізнає введені користувачем символи з точністю не меншою ніж 75%, зокрема має передбачатися перевірка того, що поля параметрів нейронної мережі повинні бути заповненими.

3.2 Вибір мови програмування

C++ – надзвичайно потужна мова, що містить засоби створення ефективних програм практично будь-якого призначення, від низькорівневих утиліт та драйверів до складних програмних комплексів різного призначення [15].

До переваг можна віднести:

- підтримуються різні стилі та технології програмування, включаючи традиційне директивне програмування, ООП, узагальнене програмування, метапрограмування (шаблони, макроси);
- стандартна бібліотека STL;
- можливість роботи на низькому рівні з пам'яттю, адресами, портами;

- можливість створення узагальнених контейнерів та алгоритмів для різних типів даних, їх спеціалізація та обчислення на етапі компіляції, використовуючи шаблони;

- кросплатформність. Доступні компілятори для великої кількості платформ, мовою C++ розробляють програми для різних платформ і систем;

- ефективність. Мова спроектована так, щоб дати програмісту максимальний контроль над усіма аспектами структури та порядку виконання програми.

До недоліків можна віднести:

- відсутність рефлексії;
- відсутність автоматичного контролю за пам'яттю;
- синтаксис, що провокує помилки;
- препроцесор, успадкований від C, дуже примітивний;
- погана підтримка модульності (по суті, у класичному C модульність на рівні мови відсутня, її забезпечення перекладено на компонувальник). Підключення інтерфейсу зовнішнього модуля через препроцесорну вставку заголовного файлу `#include` серйозно уповільнює компіляцію при підключенні великої кількості модулів (оскільки результуючий файл, який обробляється компілятором, виявляється дуже великий);

- відсутність системи збирання сміття. З іншого боку, у C++ є достатньо засобів (класи з конструкторами та деструкторами, стандартні шаблони, передача параметрів за посиланням), що дозволяють виключити використання небезпечних покажчиків. Тим не менш, відсутність вбудованого складання сміття дозволяє користувачеві вибрати стратегію управління ресурсами.

У табл. 3.1 наведено порівняння обраної мови програмування із іншими найпопулярнішими мовами програмування.

Таблиця 3.1 – Порівняння мов програмування

Критерії порівняння	Мова програмування		
	C++	C#	Python
Кількість сторонніх бібліотек	Середня	Среденя	Велика
Типізація	Статична	Статична	Динамічна
Швидкодія	Висока	Середня	Низька
Складність вивчення	Висока	Середня	Низька
Швидкість розробки	Середня	Висока	Висока
Збирання сміття	Немає	Є	Є
Метод виконання коду	Компіляція у машинний код	Компіляція у байт-код	Інтерпретація

Як видно з таблиці не дивлячись на те, що мова програмування C++ виникла вже достатньо давно, вона і зараз достатньо популярна та активно продовжує розвиватися.

За результатами можна зробити висновок, що велике практичне коло використання C++ для розробки застосунків значно відрізняє її між інших аналогів. Важливим параметром у розробці ігрових додатків є швидкодія. Саме тому для розробки ПЗ було обрано саме C++.

У якості середовища розробки була обрана Visual Studio 2019. Вибір засновується на її сумісності з UE4 та додатком VisualAssist, який значно спрощує розробку додатків.

3.3 Вимоги до технічних засобів

Комп'ютер користувача має відповідати мінімальним вимогам ігрового рушія, а саме:

- операційна система – Windows 7/8/10/11 x64;

- процесор – 4 ядра з тактовою частотою – 2,5 ГГц;
- оперативна пам'ять – 8Гб;
- відеокарта – з підтримкою DirectX 11;
- вільний дисковий простір – не менше 30 Гб;
- ігровий рушій UE4 версії 4.26.

3.4 Проєктування розробки

3.4.1 Архітектура ПЗ

Для реалізації ПЗ була обрана модульна система: програмний модуль і модуль графічного інтерфейсу користувача. Програмний модуль повинен виконувати дві основні задачі:

- навчання нейронної мережі;
- збереження результатів.

Модуль графічного інтерфейсу відповідає за взаємодію користувача з ПЗ. Він має спростувати процес генерації навчальної вибірки та демонструвати процес навчання.

Така організація архітектури застосунку спростить розробку, забезпечить розподіл функціоналу між окремими класами, що гарантує легкість підтримки та розширення функціоналу застосунку, особливо при виході нових версій UE4.

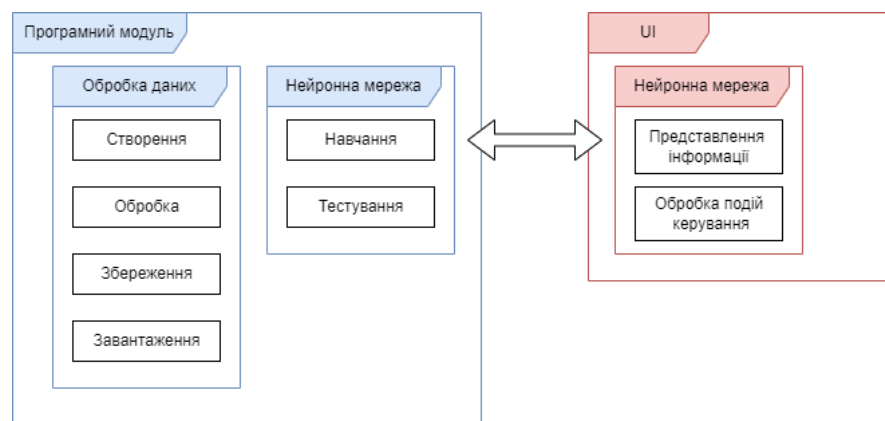


Рисунок 3.1 – Модульне представлення ПЗ

3.4.2 Діаграма прецедентів

Діаграма прецедентів відображає відносини між акторами та прецедентами системи. Прецедент визначає поведінку, демонстроване системою з метою отримання значущого результату однієї чи більше акторів.

Прецедент відповідає окремому сервісу системи, що визначає один з варіантів її використання та описує типовий спосіб взаємодії користувача з системою. Діаграма прецедентів для системи представлена на рис. 3.2.

В рамках системи передбачений один актор – користувач, який може:

а) Редагувати навчальну вибірку:

- 1) імпортувати навчальну вибірку;
- 2) створити навчальну вибірку;
- 3) видалити навчальну вибірку;

б) редагувати НМ:

- 1) навчити мережу на навчальній вибірці зображень;
- 2) редагувати параметри НМ, а саме – кількість нейронів у прихованому шарі;
- 3) редагувати параметри навчання: швидкість, кількість циклів, автоматичне навчання та допустиму точність;
- 4) завантажити НМ;

в) тестувати точність розпізнання навченої НМ;

г) розпізнати символ за допомогою навченої НМ.

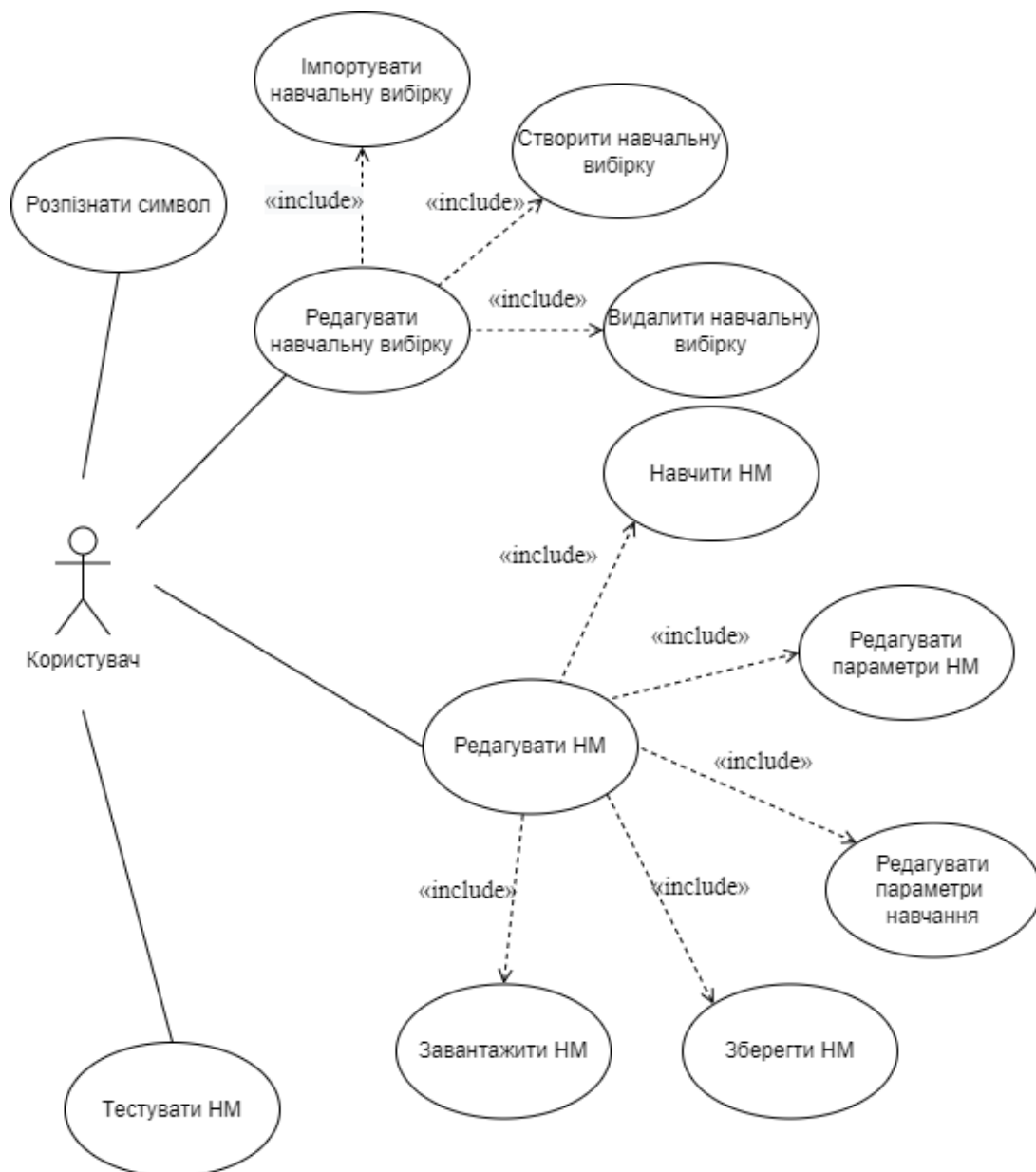


Рисунок 3.2 – Діаграма прецедентів

3.4.3 Топологія НМ

Для задачі розпізнання символів було обрано архітектуру НМ у вигляді багатоварового перцептронів з одним прихованим шаром (рис. 3.3).

Топологія НМ:

- вхідний шар: 784 нейрони – зображення символу розміром 28x28 пікселів;
- прихований шар: 100-300 нейронів;

– вихідний шар: 3-30 нейронів – номер розпізнаного символу.

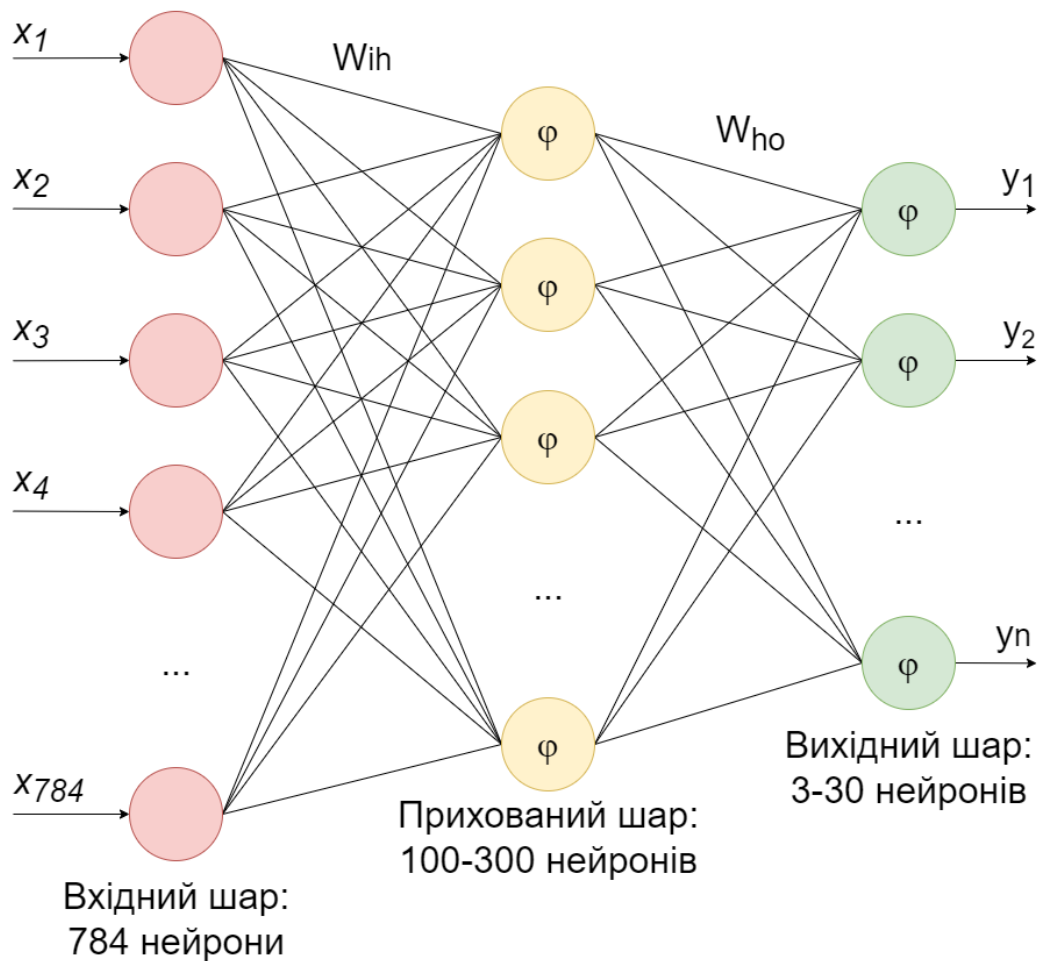


Рисунок 3.3 – Топологія НМ

3.4.4 Формування навчальної вибірки

Для навчання НМ необхідна навчальна вибірка зображень символів розміром 28x28 пікселів. На кожен символ необхідно від п'яти до двадцяти зображень. Для зручності необхідно реалізувати підсистему, за допомогою якої можна буде формувати навчальну вибірку у редакторі. Для цього у редакторі буде розташоване полотно, на якому можна малювати зображення символу у великій роздільній здатності, яке потім буде формуватись під необхідний розмір. Зображення символів зберігаються у каталозі проєкту у форматі PNG. Також можливий імпорт зображень.

3.5 Висновки за розділом 3

У результаті аналізу засобів розробки була обрана мова програмування C++, завдяки його швидкості роботи і сумісності з UE4, а у якості середовища розробки було обране середовище розробки Visual Studio 2019, також через його сумісність з ігровим рушієм. Сформоване модульне представлення ПЗ та визначені функціональні вимоги до програми, які були зображені на діаграмі прецедентів. Розглянута топологія НМ на основі багатошарового персептрону з одним прихованим шаром.

4 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

4.1 Програмне забезпечення

Для реалізації функціоналу було розроблено ПЗ з набором модулів, запропонованим у розділі 3. ПЗ складається з десяти класів, серед яких можна виділити наступні:

- Matrix;
- NeuralNetwork;
- TrainingDataset;
- NetworkTariningAsyncTask;
- Data;
- Plugin;
- PluginEditor.

Клас Matrix призначений для зберігання даних ваг у вигляді двовимірного масиву дійсних чисел, а також проведення операцій над ними.

Клас NeuralNetwork призначений для зберігання даних про НМ і методи тренування НМ.

Клас TrainingDataset призначений для зберігання інформації про параметри навчання НМ і навчальну вибірку.

У класі NetworkTariningAsyncTask реалізовано метод навчання НМ у фоновому потоці та відображення процесу у UI.

Клас Data призначений для зберігання даних про профілі нейронних мереж та відповідає за роботу з ними.

Клас Plugin централізує в собі роботу всієї системи та відповідний за використання вже навчених НМ в ігровому процесі.

У класі PluginEditor реалізовано взаємодію з користувачем та вбудовуванням у ігровий рушій.

4.1.1 Клас Matrix

Клас Matrix відповідний за представлення ваг у вигляді двовимірних масивів та проведенням операцій над ними. Також у ньому описані функції активації нейронів. За допомогою об'єктів класу Matrix у ПЗ реалізується нейрони та операції над ними.

До властивостей класу відносяться:

- uint32 NumRows;
- uint32 NumColumns;
- TArray<FSRRowItem> R.

Серед методів класу можна виділити:

- Matrix CompWiseMultiply(const Matrix& Other);
- Matrix ActivationOperation(EActivationFunc InActivationFunc);
- Matrix ToSoftMax();
- void SetFromData(uint32 rows, uint32 cols, const TArray<float>& InData);
- float SigmoidFunc(const float& Z);
- float ReLUFunc(const float& Z);
- float TanHFunc(const float& Z).

4.1.2 Клас NeuralNetwork

Клас NeuralNetwork відповідний за реалізацію топології нейронної мережі описаної у вигляді багатошарового персептрону з одним прихованим шаром.

До властивостей класу відносяться:

- uint32 InputNodes;
- uint32 HiddenNodes;
- uint32 OutputNodes;

- float LearningRate;
- FSRDMatrix wih;
- FSRDMatrix who;
- bool bIsTrained.

Серед методів класу можна виділити:

- void Train(const TArray<float>& InputList, const TArray<float>& OutputList);
- FSRDMatrix Query(const TArray<float>& InputList) const;
- static int32 GetQueryResult(const FSRNeuralNetwork& Neural, const TArray<float>& QueryData, int32 AnswerIdx, float AcceptableAnswerSize = 0.5, float DeltaBestAnswers = 0.97).

4.1.3 Клас TrainingDataset

Клас TrainingDataset призначений для зберігання інформації про параметри навчання НМ і навчальну вибірку.

До властивостей класу відносяться:

- TArray<TArray<float>> Inputs;
- int32 Outputs;
- int32 Answer;
- TArray<float> ExpectedOutput.

До методів класу відноситься лише конструктор, призначений для ініціалізації властивостей класу.

4.1.4 Клас NetworkTariningAsyncTask

У класі NetworkTariningAsyncTask реалізовано метод навчання НМ у фоновому потоці та відображення процесу у UI.

До властивостей класу відносяться:

- FSRNeuralNetwork& NeuralItem;

- TArray<FSRTrainingDataSet> TrainingsSet;
- int32 Outputs;
- int32 AllImagesCount;
- uint32 Inputs;
- uint32 Hidden;
- float LearningRate;
- int32 EpochsLimit;
- float AcceptableAccuracy;
- float DeltaBestAnswers;
- FTrainingTaskCompleteDelegate TrainingTaskComplete;
- FTrainingTaskStopDelegate TrainingTaskStop;
- static FThreadSafeBool bShouldStopSymbolTraining;
- static float Progress;
- static int32 CurrentEpochs;
- static TArray<float> SymbolsScores.

До методів класу відноситься DoWork(), який призначений для навчання НМ у фоновому процесі.

4.1.5 Клас Data

Клас Data призначений для зберігання даних про профілі нейронних мереж та відповідає за роботу з ними.

До властивостей класу відносяться:

- FString CurrentProfile;
- TMap<FString, FSRNeuralNetwork> NeuralProfiles;
- TMap<FString, FSRDrawLayerWrapper> DrawLayers.

Серед методів класу можна виділити:

- bool ReadDrawLayers(const FString InProfile, TArray<FSRDrawingLayer>& OutDrawLayers);

– void WriteDrawLayers(const FString InProfile, const TArray<FSRDrawingLayer>& InDrawLayers).

4.1.6 Клас Plugin

Клас Plugin централізує в собі роботу всієї системи та відповідний за використання вже навчених НМ в ігровому процесі.

До властивостей класу відносяться:

- FSRNeuralNetwork NeuralNetwork;
- bool bIsLoaded;
- int32 NeuralTextureSize;
- bool bIsDrawing;
- bool bAddNewDrawSpot.

Серед методів класу можна виділити:

- void BeginDrawing();
- void Draw(FVector2D InDrawPosition);
- void EndDrawing();
- void ClearCanvas();
- void SaveNeuralProfile(const FString InProfile, FSRNeuralNetwork& InNeuralNetwork);
- bool LoadNeuralNetworkFromSRData(FSRNeuralNetwork& NeuralData);
- bool SelectProfile(const FString InProfile, bool bAddEmptyIfNotFound = false, bool bShouldSave = true);
- bool DeleteProfile(const FString InProfile);
- int32 GetMostAccurateSymbol(float AccuracyTreshold = 0.3f) const;
- bool TestDrawing(int32 ExpectedSymbol = 0, float AccuracyTreshold = 0.3f, bool bCompareToMostAccurateSymbol = true) const.

Діаграму основних класів застосунку наведено на рис. 4.1.

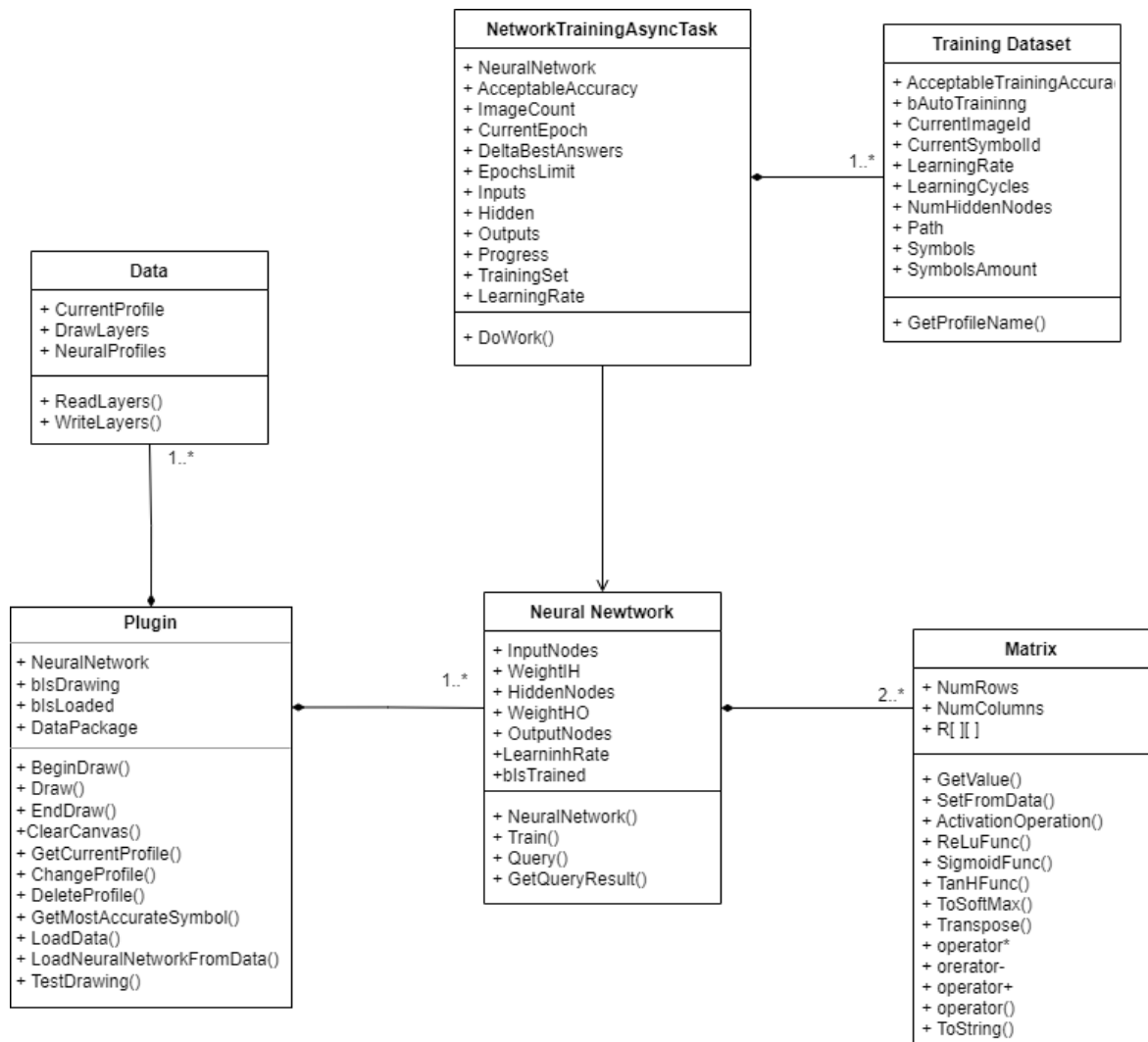


Рисунок 4.1 – Діаграма класів

4.2 Висновки за розділом 4

У розділі описаний поетапний процес розробки програми в середовищі Visual Studio 2019. Описана технологія розробки з використанням мови програмування C++. Наведено опис розроблених програмних класів, досліджено їх методи та властивості. Розроблено ПЗ для реалізації та дослідження впливу активаційної функції нейронів прихованого шару на швидкість навчання і точність роботи НМ.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ

5.1 Призначення програми

Плагін призначений для створення нейронних мереж, які розпізнають користувацькі символи, включаючи створення навчальної вибірки, створення, навчання і тестування нейронної мережі.

Під час тестування будуть порівнюватися функції активації нейронів прихованого шару та метод визначення витрат для корегування ваг.

5.2 Тестовий набір даних

Для тестування розробленого ПЗ буде розглядатися динаміка зміни значення функції витрат і точності розпізнання НМ від функції витрат та активаційної функції нейронів прихованого шару.

Будуть розглядатися такі функції активації:

- логістична (сігмоїда);
- гіперболічний тангенс;
- випрямлена лінійна.

Витрати будуть розраховуватися за допомогою методів середньоквадратичного відхилення та перехресної ентропії.

У якості тестової вибірки було сформовано набір з десяти символів по двадцять зображень у кожному. Навчання нейронної мережі буде проводитися протягом 100 епох.

5.3 Результати тестування

Кінцеві результати експерименту наведено у табл. 5.1.

Таблиця 5.1 – Порівняння функцій активації та методів

Метод визначення витрат:	Метод середньоквадратичного відхилення			Метод перехресної ентропії		
Функція активації:	Витрати	Точність	Час, сек.	Витрати	Точність	Час, сек.
Логістична	0,0645	60,65%	147	0,2945	91,89%	134
Гіперболічний тангенс	0,0195	89,24%	133	0,163	95,25%	121
Випрямлена лінійна	0,0197	89,42%	124	0,1186	96,30%	112

Результати експерименту було сформовано у виді графіків для зручності порівняння. На рис. 5.1 зображено динаміку зміни значення витрат методом середньоквадратичного відхилення.

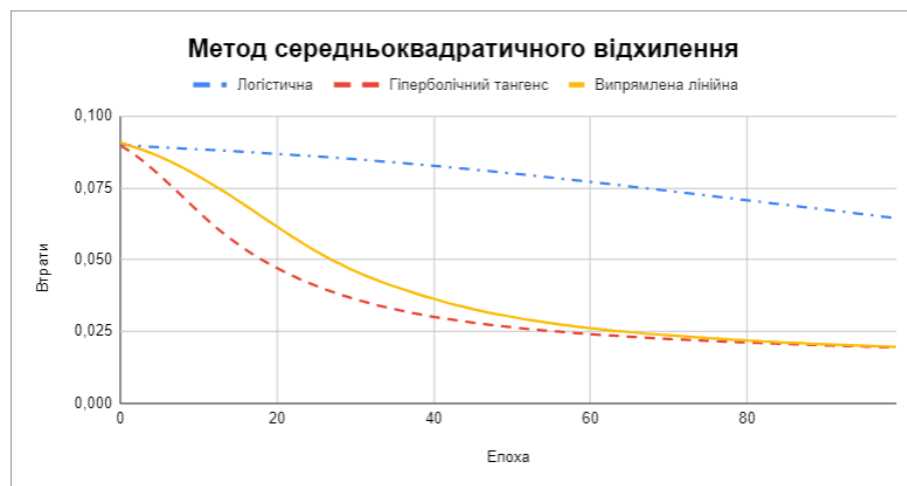


Рисунок 5.1 – Динаміка витрат, обчислених методом середньоквадратичного відхилення

На рис. 5.2 зображено динаміку зміни точності методом середньоквадратичного відхилення для кожної з функцій активації.

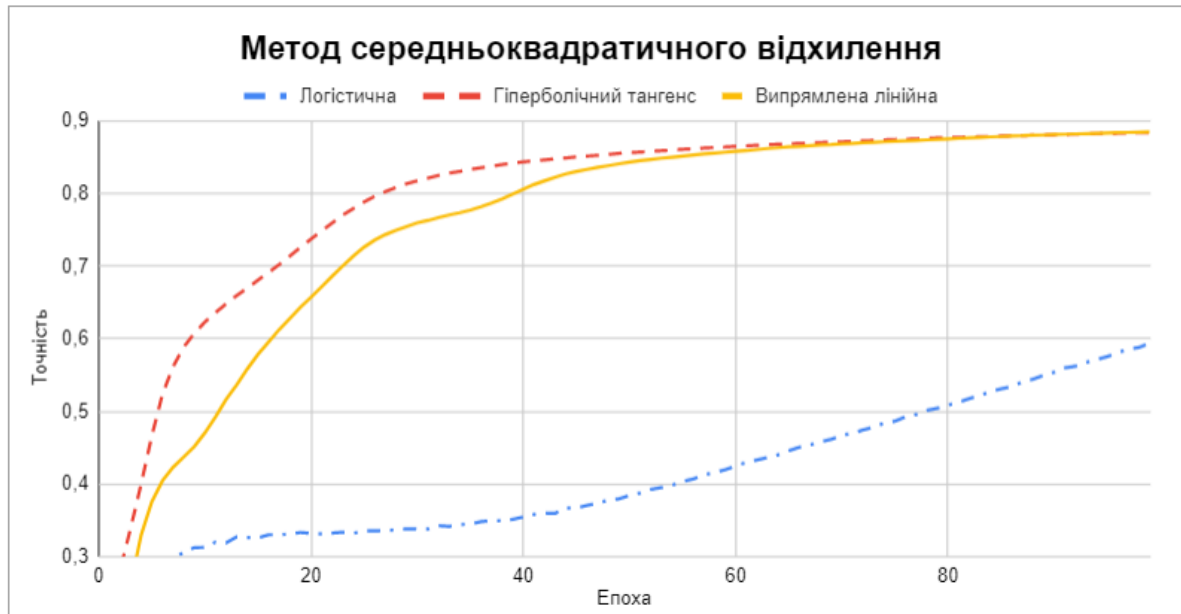


Рисунок 5.2 – Динаміка точності при використанні методу середньо квадратичного відхилення

На рис. 5.3 зображено динаміку зміни значення витрат методом перехресної ентропії.

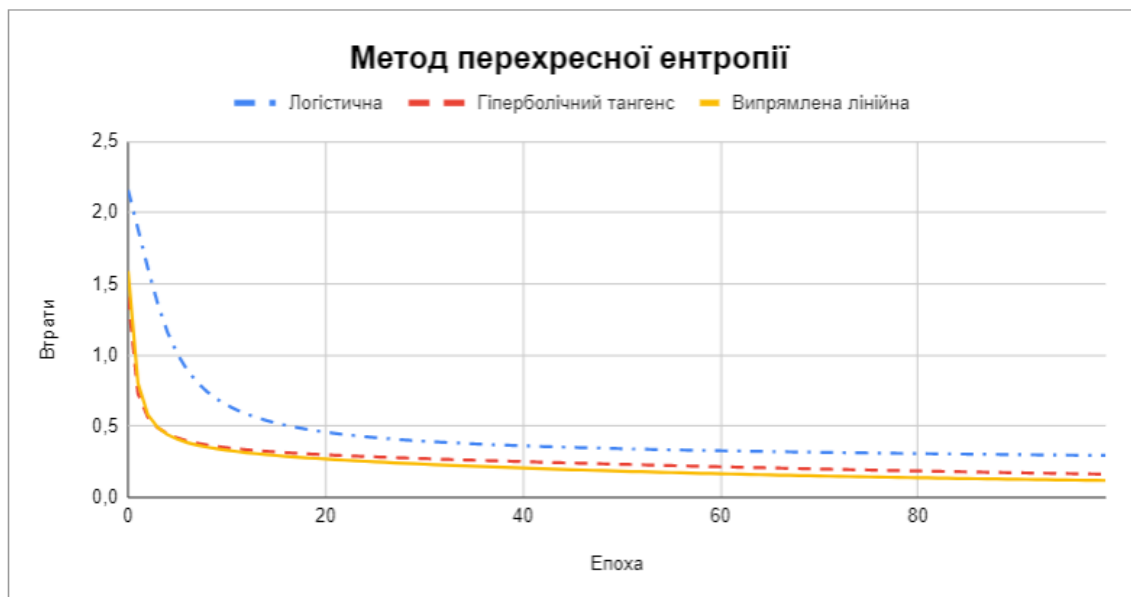


Рисунок 5.3 – Динаміка витрат, обчислених методом перехресної ентропії

На рис. 5.4 зображено динаміку зміни точності НМ методом перехресної ентропії.

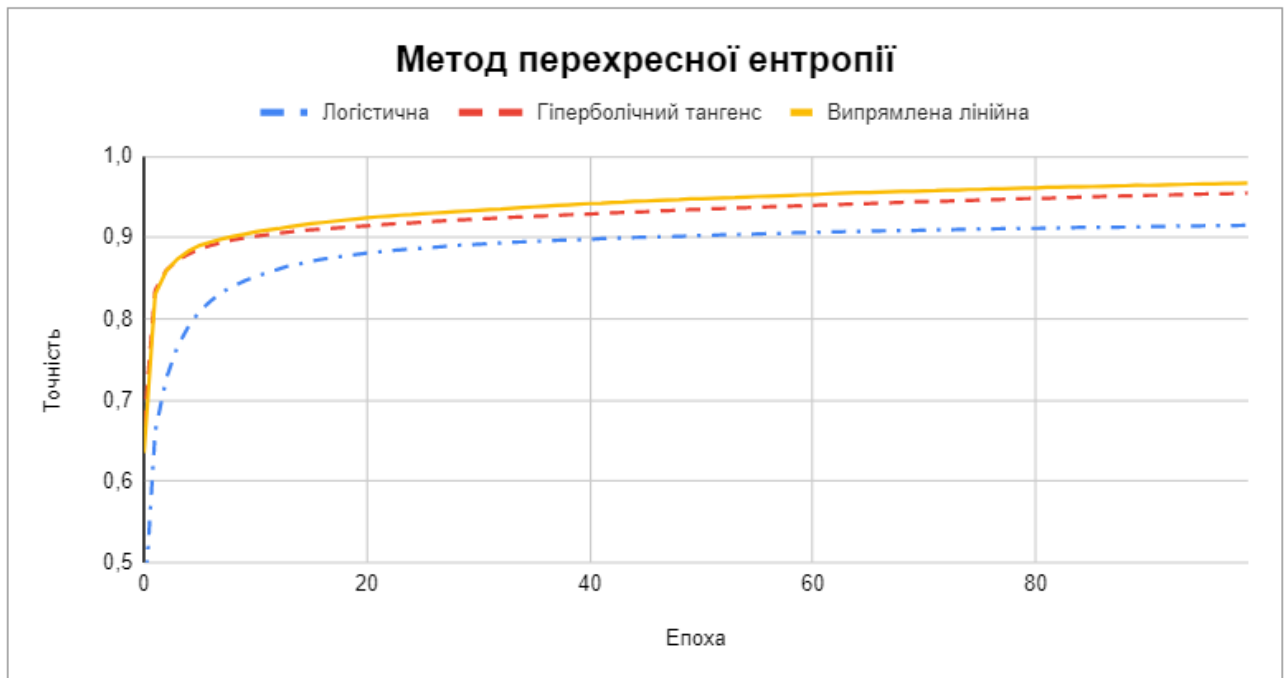


Рисунок 5.4 – Динаміка точності при використанні методу перехресної ентропії

При використанні методу середньоквадратичного відхилення у якості функції визначення витрат, гіперболічний тангенс і випрямлена лінійна функція активації показують себе майже однаково, на відміну від логістичної функції активації, якій потрібно у два рази більша кількість епох, щоб наблизитися до їх значень точності розпізнання.

При порівнянні рис. 5.2 і рис. 5.4 видно, що визначення функції витрат методом перехресної ентропії значно прискорює навчання нейронної мережі в порівнянні з методом середньоквадратичного відхилення. Гіперболічний тангенс і випрямлена лінійна функція активації показують себе майже однаково, але друга має нижчу складність обчислення.

Найкращих результатів точності та швидкості навчання вдалося досягти використовуючи метод перехресної ентропії для визначення витрат і випрямлену лінійну функцію активації нейронів прихованого шару.

5.4 Тест кейси для розробленого ПЗ

Тестування ПЗ було проведено для версії UE4 4.26. У табл. 5.2. наведено результати тестування.

Таблиця 5.2 – Тест кейс тестування

Опис кейсу	Очікуваний результат	Результат
Підключення плагіну до ігрового рушія	Плагін створить необхідні файли у каталозі ігрового проєкту. У редакторі з'явиться кнопка виклику плагіну	Пройдено
Запуск плагіну з панелі редактору	Відкриється вікно плагіну. Завантажаться існуючі профілі НМ	Пройдено
Створення нового профілю	Створиться порожній профіль НМ з заданою кількістю символів та зображень до них	Пройдено
Створення зображення символу	Зображення конвертується у потрібний розмір та збережеться у каталозі проєкту	Пройдено
Навчання НМ	Виконання необхідних обчислень для навчання НМ для розпізнання заданих символів.	Пройдено
Тестування навченої НМ	Навчена НМ розпізнає символ на полотні з відображенням максимального значення вихідного нейрону	Пройдено
Виклик НМ у ігровому процесі	При розпізнанні правильного символу в процесі гри відбудеться виклик ігрової логіки	Пройдено

5.5 Опис роботи з програмою

Спочатку необхідно підключити плагін до рушія Unreal Engine 4. Після цього всередині рушія необхідно натиснути на кнопку виклику плагіну (рис. 5.5).

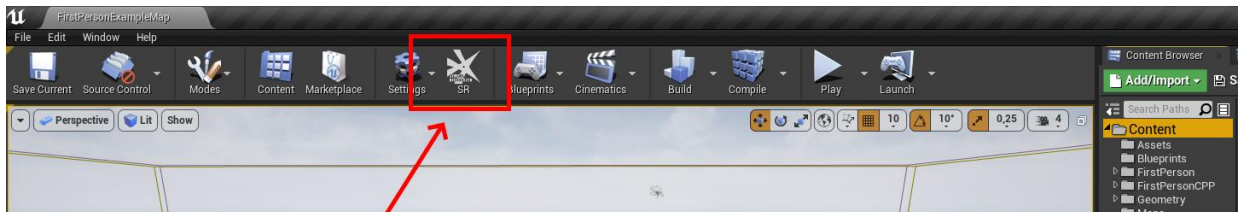


Рисунок 5.5 – Відкриття вікна редактору

Відкриється вікно редактору. Далі необхідно створити новий профіль для збереження символів. Для цього потрібно натиснути кнопку зображену на рисунку 5.6. Необхідно налаштувати кількість символів та кількість прикладів зображень кожного символу. Для коректної роботи плагіну рекомендується кількість символів не більше тридцяти. При перевищенні цього ліміту значно підвищується час навчання. А кількість зображень для кожного символу повинна становити від п'яти до двадцяти.

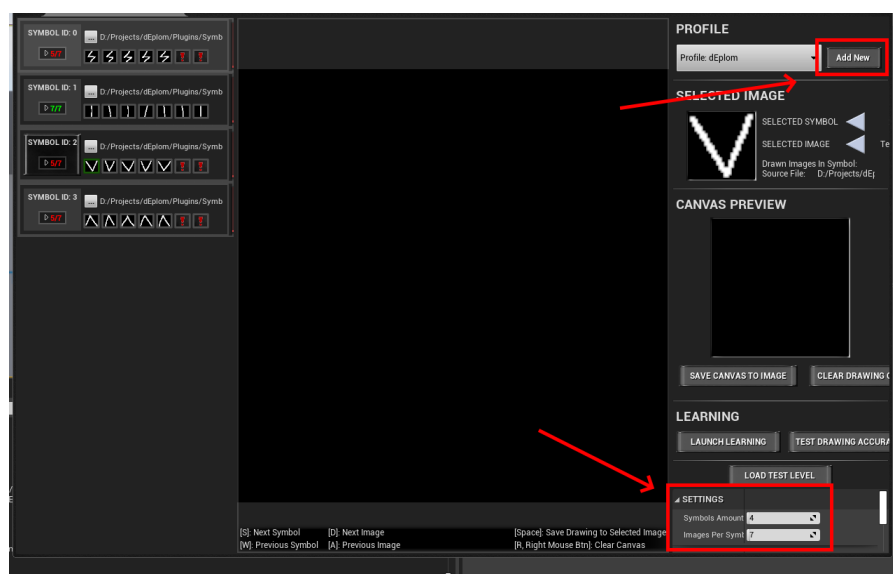


Рисунок 5.6 – Додавання нового профілю

Потрібно створити навчальну вибірку. Для цього необхідно намалювати зображення кожного символу обране число разів. Це можна зробити намалювавши їх прямо у редакторі чи імпортувати готові рисунки. Процес додавання рисунку з редактору зображено на рисунку 5.7. При додаванні через вбудований редактор малювати символ можна у високій якості. Він буде автоматично підігнаний під необхідні розміри. Процедуру необхідно повторити для кожного символу.

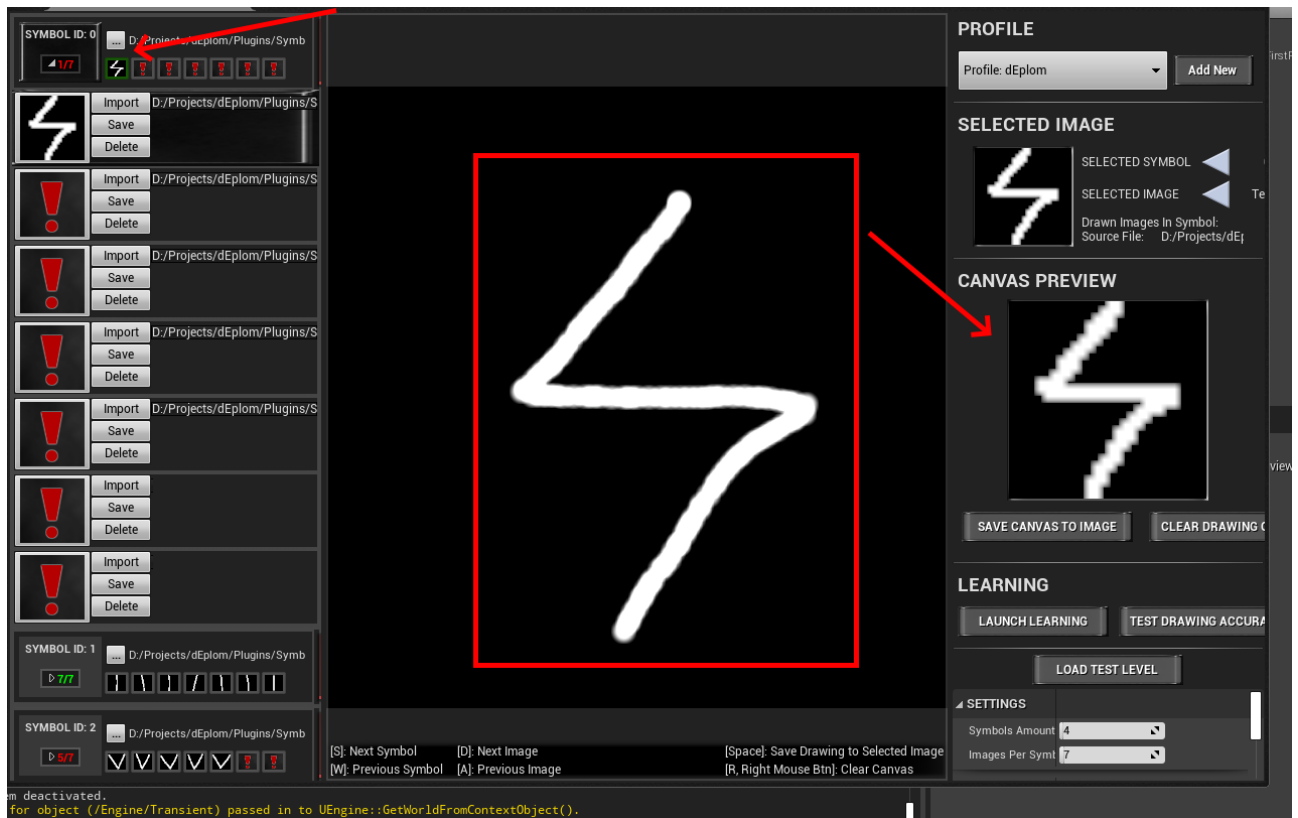


Рисунок 5.7 – Додавання зображення символу.

Після додавання всіх зображень для кожного символу необхідно налаштувати параметри нейронної системи (рис.5.8).

LEARNING

LAUNCH LEARNING TEST DRAWING ACCURACY

LOAD TEST LEVEL

SETTINGS

Symbols Amount	4
Images Per Symbol	7
Learning Rate	0,15
Hidden Nodes	100
Auto Training	☒
Learning Cycles	200
Acceptable Training Accuracy	0,95
Delta Two Best Outcomes	0,9

Рисунок 5.8 – Налаштування нейронної мережі

Швидкість навчання визначає розмір кроку градієнтного спуску. Надто мале значення вимагатиме надто великої кількості ітерацій, що значно збільшить час навчання, а надто велике значення може пройти повз мінімум функції. Рекомендована величина: від 0,1 до 0,25.

Кількість нейронів у прихованому шарі впливає як на точність нейронної мережі так і на швидкість її навчання. Залежить від кількості символів у профілі – для більшої кількості символів потрібно більша кількість нейронів. Рекомендована величина: від 100 до 300.

Навальні цикли визначають кількість епох, які навчатиметься нейронна мережа. За допомогою авто навчання можна автоматично визначати необхідну кількість епох – навчання нейронної мережі завершиться при досягненні допустимої точності навчання.

Після налаштування всіх параметрів можна запускати процес навчання (рис. 5.9).

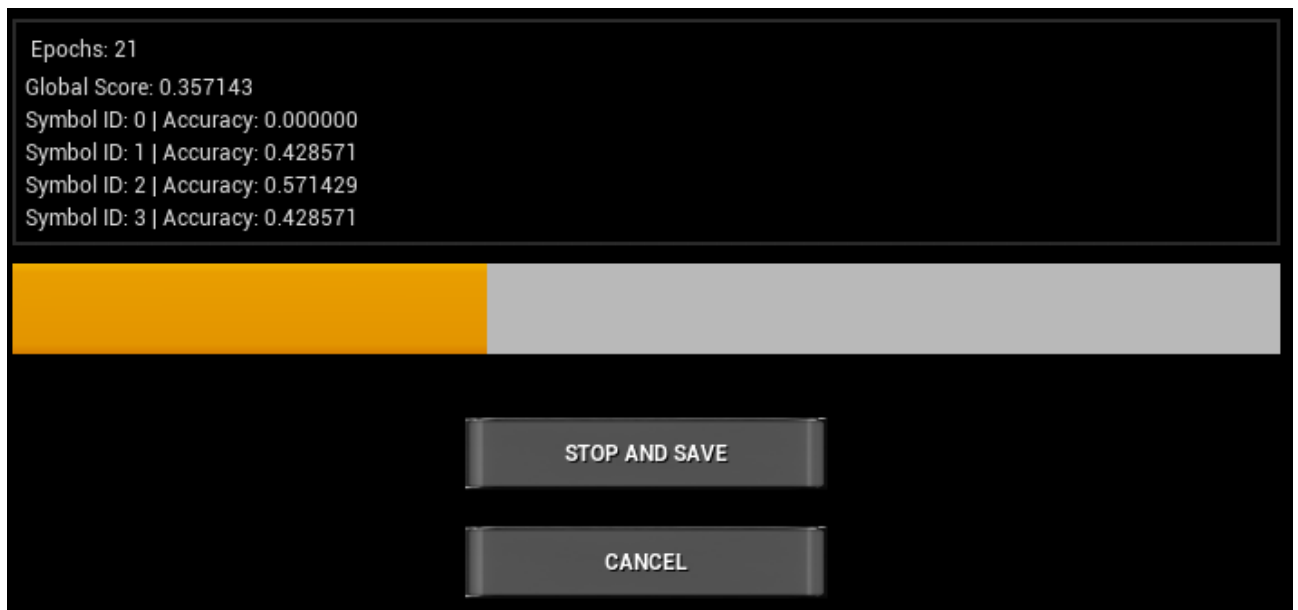


Рисунок 5.9 – Процес навчання нейронної мережі

Під час навчання відображається кількість епох, середня точність нейронної мережі та точність для кожного символу.

Після завершення навчання є можливість протестувати нейронну мережу (рис. 5.10).

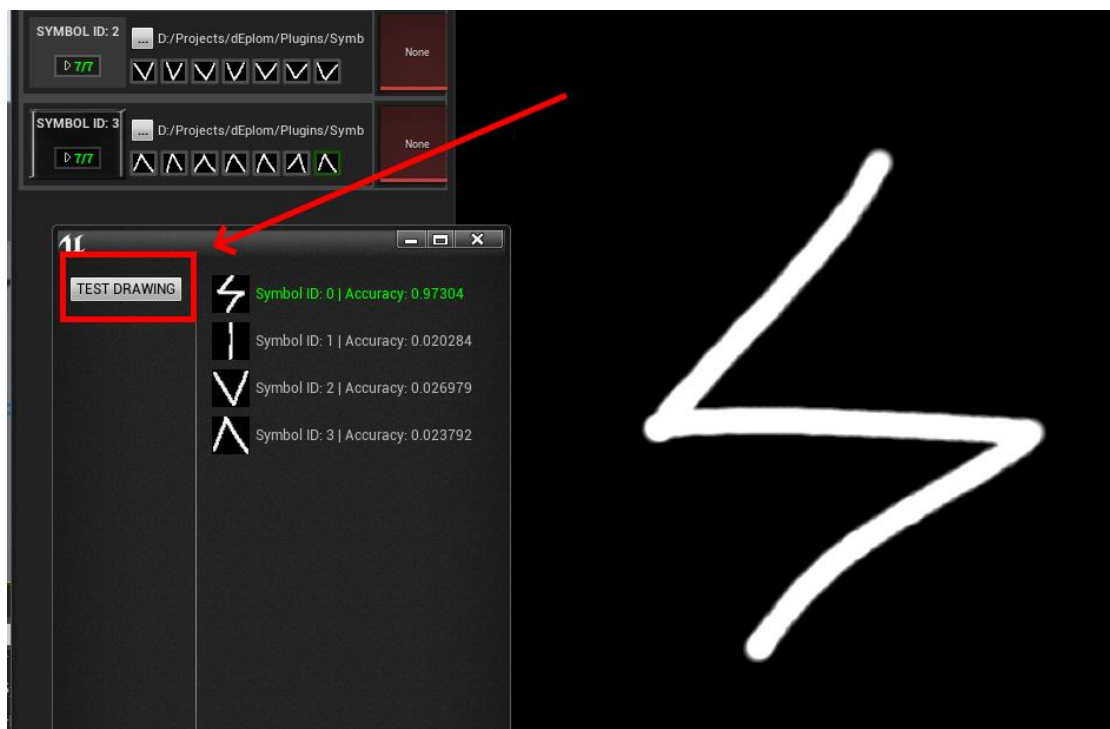


Рисунок 5.10 – Тестування нейронної мережі

Для тестування необхідно намалювати символ для розпізнання, після чого натиснути на кнопку зображену на рис. 5.6. Праворуч від неї розташовуються результати. Зеленим кольором підсвічено відповідь нейронної мережі її точність.

Для використання нейронної мережі всередині гри необхідно викликати метод Draw() для передачі малюнку (рис. 5.11).

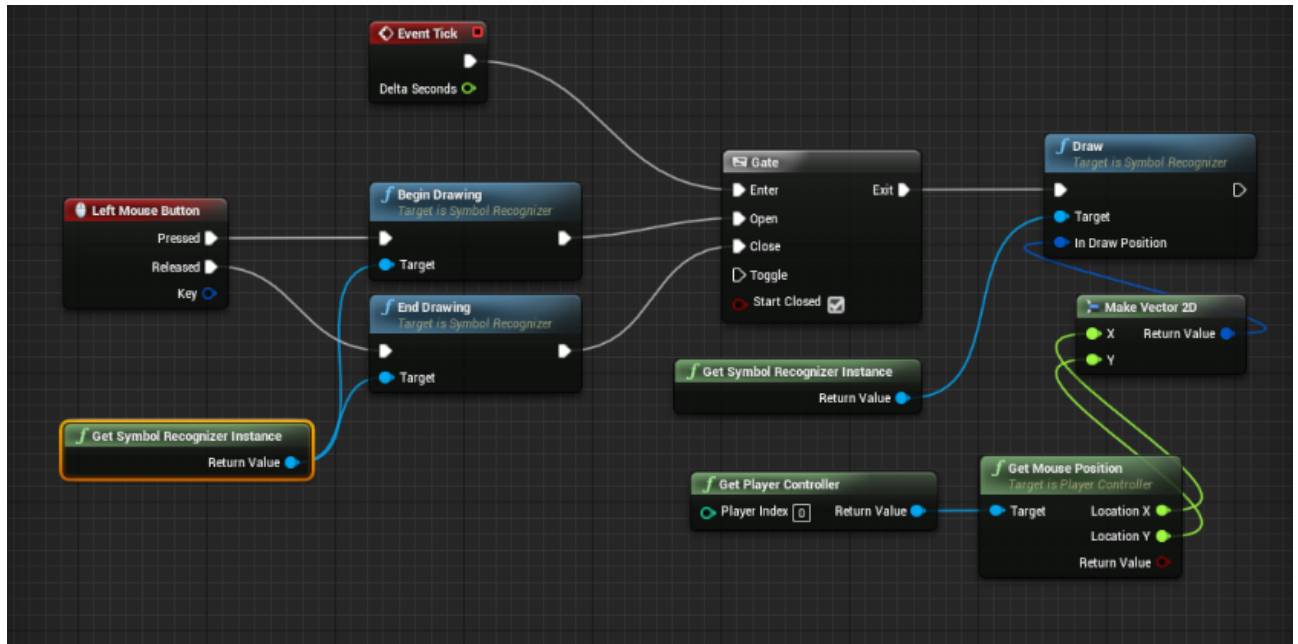


Рисунок 5.11 – Відображення процесу передачі малюнку до нейронної мережі

А для отримання результату викликати метод GetMostAccurateSymbol(), який повертає індекс символу з найбільшою точністю (рис. 5.12).



Рисунок 5.12 – GetMostAccurateSymbol()

За допомогою цих двох функцій можна реалізувати виклик ігрової логіки, керування чи застосування здібностей персонажу.

5.6 Висновок за розділом 5

Проведено експериментальну оцінку ефективності методів визначення витрат і функцій активації нейронів прихованого шару. Метод перехресної ентропії разом з випрямленою лінійною функцією активації показав найкращі результати при задачі багатокласової класифікації, тому в ПЗ реалізовано саме цей варіант.

ВИСНОВКИ

З метою підвищення рівня автоматизації, точності та швидкості навчання в роботі досліджено актуальну проблему розпізнання рукописних символів шляхом створення програмної системи на основі методів і моделей штучного інтелекту.

Проведено аналіз предметної області, у результаті якого було обрано архітектуру НМ. У результаті огляду існуючих програмних засобів була визначена недоцільність їх використання у даній роботі. Було визначено функціональні вимоги до програми.

Розглянуто метод зворотного розповсюдження похибки для навчання НМ. Також розглянуто функції активації нейронів та методи визначення витрат.

Проведено проектування архітектури ПЗ для реалізації. Було виконано порівняння засобів реалізації ПЗ, в результаті чого було обрано ігровий рушій UE4, мову програмування C++ та середовище розробки Visual Studio.

Проведено експериментальну оцінку ефективності методів визначення витрат і функцій активації нейронів прихованого шару. Метод перехресної ентропії разом з випрямленою лінійною функцією активації показав найкращі результати при задачі багатокласової класифікації, тому в ПЗ реалізовано саме цей варіант.

Створено програмний застосунок, який у формі плагіна підключається до ігрового рушія Unreal Engine 4 і вирішує задачу створення та навчання нейронної мережі. Розроблено програмну систему для розпізнавання рукописних символів, яка автоматизує та прискорює процес створення та навчання нейронної мережі.

Галузь використання – розробка ігор для комп'ютерних і мобільних платформ з символьним керуванням.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Newzoo Global Games Market Report 2021 [Electronic resource] – Access mode: <https://newzoo.com/insights/trend-reports/newzoo-global-games-market-report-2021-free-version/>.
2. В App Store публікується по 800 ігор ежедневно [Електронний ресурс] – Режим доступу: <http://app2top.ru/industry/v-app-store-publikuetsya-po-800-igrezhednevno-91573.html>.
3. Гафаров, Ф.М Искусственные нейронные сети и приложения: учеб. пособие / Ф.М. Гафаров, А.Ф. Галимянов. – Казань: Изд-во Казан. ун-та, 2018. – 121 с.
4. Нейронные сети, перцептрон [Электронный ресурс] – Режим доступа: https://neerc.ifmo.ru/wiki/index.php?title=Нейронные_сети,_перцептрон.
5. Субботін, С.О. Нейронні мережі : навч. посіб. / С.О. Субботін, А.О. Олійник; за ред. С.О. Субботіна. – Запоріжжя : ЗНТУ, 2014. – 132 с.
6. Бураков, М.В. Нейронные сети и нейроконтроллеры: учебное пособие / М.В. Бураков. – СПб.: ГУАП, 2013. – 284 с.
7. Основы глубокого обучения Power Scores 2018 [Электронный ресурс] – Режим доступа: <https://www.machinelearningmastery.ru/deep-learning-framework-power-scores-2018-23607ddf297a/>.
8. TensorFlow C++ API Reference [Electronic resource] – Access mode: https://www.tensorflow.org/api_docs/cc/.
9. Unreal Engine [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Unreal_Engine.
10. Programming with C++ [Electronic resource] – Access mode: <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/ProgrammingWithCPP/>.
11. TensorFlow-UE4 [Electronic resource] – Access mode: <https://github.com/getnamo/tensorflow-ue4>.

12. Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks [Electronic resource] – Access mode: <https://arxiv.org/abs/1312.6082>.
13. Крижевський, А. Imagenet classification with deep convolutional neural networks / А. Крижевський, І. Суткевер, Г.Є. Хінтон. – 2012. – Р. 1097–1105 с.
14. Градієнтний спуск [Електронний ресурс]. – Режим доступу https://uk.wikipedia.org/wiki/Градієнтний_спуск.
15. Что такое TensorFlow? [Электронный ресурс] – Режим доступа: <https://coderlessons.com/tutorials/mashinnoe-obuchenie/uchebnik-tensorflow/1-chto-takoe-tensorflow>.
16. Активационная функция нейрона [Электронный ресурс] – Режим доступа: <https://wiki.loginom.ru/articles/activation-function.html>.
17. Multilayer perceptron architecture [Electronic resource] – Access mode: <https://www.sanfransentinel.com/neuralnetworkperceptrone.html>.

ДОДАТОК А
Технічне завдання

A.1 Вступ

A.1.2 Призначення та область застосування

У роботі необхідно виконати розробку програмного забезпечення створення нейронних мереж для розпізнавання рукописних символів для підвищення автоматизованості даного процесу.

A.2 Підстави для розробки

Розробка виконується у відповідності з завданням, яке було визначено на кваліфікаційну роботу магістра за темою “Розробка і дослідження системи розпізнавання символів сенсорного вводу з використанням машинного навчання”, що затверджена наказом № 435 від 5 листопада 2021 р. по Національному університету «Запорізька політехніка».

A.3 Призначення розробки

Програма, що розробляється, призначена для створення нейронної мережі, включаючи попередні налаштування нейронної мережі, інструменти для створення вхідного масиву символів для розпізнавання, тренування, тестування та використання всередині ігрового рушія, що має в результаті спростити і автоматизувати більшу частину процесу.

A.4 Основні вимоги до програми

A.4.1 Вимоги до функціональних характеристик

Програма має реалізовувати наступні функціональні характеристики:

- створення і редагування даних профілів з символами;
- створення і редагування даних символів всередині профілю;
- налаштування параметрів чутливості нейронної мережі;

- відображення процесу навчання нейронної мережі;
- відображення результатів тестування нейронної мережі;
- використання результатів роботи нейронної мережі під час гри (виклик ігрової логіки).

А.4.2 Вимоги до надійності

Програмне забезпечення має створювати нейронну мережу, яка коректно розпізнає введені користувачем символи з точністю не меншою ніж 75%, зокрема має передбачатися перевірка того, що поля параметрів нейронної мережі повинні бути заповненими.

А.4.3 Умови експлуатації

Для експлуатації програмного забезпечення має бути встановлено і запуснено ігровий рушій Unreal Engine 4 версії 4.26.

А.4.4 Вимоги до складу та параметрів технічних засобів

Комп'ютер користувача має відповідати мінімальним вимогам ігрового рушія, а саме:

- операційна система – Windows 7/8/10/11 x64;
- процесор – 4 ядра з тактовою частотою – 2,5 ГГц;
- оперативна пам'ять – 8Гб;
- відеокарта – з підтримкою DirectX 11;
- вільний дисковий простір – не менше 30 Гб.

ДОДАТОК Б
Текст програми

Б.1 Текст файла Matrix.h

```
#pragma once
#include "SRMatrix.generated.h"

DECLARE_DELEGATE_RetVal_OneParam(float,
FMatrixCompWiseOperation, float);
DECLARE_DELEGATE_RetVal(float, FMatrixOperationDelegate);

USTRUCT()
struct SYMBOLRECOGNIZERPLUGIN_API FSRRowItem
{
    GENERATED_USTRUCT_BODY()

    UPROPERTY()
    TArray<float> C;

    FSRRowItem() { C.Empty(); };
    FSRRowItem(const TArray<float>& InData)
    {
        C = InData;
    }

    friend FArchive& operator<<(FArchive& Ar, FSRRowItem&
RowItem)
    {
        return Ar << RowItem.C;
    }
};

USTRUCT()
struct SYMBOLRECOGNIZERPLUGIN_API FSRDMatrix
{
    public:

    enum EActivationFunc
    {
        Sigmoid = 0,
        ReLU,
        TanH
    };

    GENERATED_USTRUCT_BODY()

    UPROPERTY()
    uint32 NumRows = 2;
    UPROPERTY()
    uint32 NumColumns = 2;
```

```

    //cannot use nesetd array cuz of serialization issues, so go
    for struct inside struct instead.
    UPROPERTY()
    TArray<FSRRowItem> R;

    FSRDMatrix();
    FSRDMatrix(uint32 rows, uint32 cols, float InitVal = 0);
    FSRDMatrix(uint32 rows, uint32 cols, const
FMatrixOperationDelegate& Operation);
    FSRDMatrix(uint32 rows, uint32 cols, TArray<float> InData);

    FSRDMatrix GetIdentity();
    float GetValue(uint32 row, uint32 col);
    FSRDMatrix GetTranspose();
    void Transpose();

    FString ToString() const;

    float& operator()(uint32 row, uint32 col);
    FSRDMatrix operator*(const FSRDMatrix& Other);
    FSRDMatrix operator*(const float& val);
    FSRDMatrix operator*=(const FSRDMatrix& Other);
    FSRDMatrix operator*=(const float& val);
    FSRDMatrix operator+(const FSRDMatrix& Other);
    FSRDMatrix operator+=(const FSRDMatrix& Other);
    FSRDMatrix operator-(const FSRDMatrix& Other);
    friend FSRDMatrix operator-(const float& lhs, const
FSRDMatrix& rhs);
    friend FSRDMatrix operator+(const float& lhs, const
FSRDMatrix& rhs);
    friend FSRDMatrix operator*(const FSRDMatrix& lhs, const
FSRDMatrix& rhs);
    FSRDMatrix operator--(const FSRDMatrix& Other);

    FSRDMatrix CompWiseMultiply(const FSRDMatrix& Other);
    FSRDMatrix CompWiseMultiplyReLU(const FSRDMatrix& Other);
    FSRDMatrix CompWiseOperation(const FMatrixCompWiseOperation&
Operation);
    FSRDMatrix ActivationOperation(EActivationFunc
InActivationFunc);
    FSRDMatrix ToSoftMax();

    void SetOrCreate(uint32 row, uint32 col, float InValue);
    void SetOrCreateRow(uint32 row, TArray<float>& InData);
    void SetFromData(uint32 rows, uint32 cols, const
TArray<float>& InData, int32 from = -1, int32 to = -1);

    FORCEINLINE float SigmoidFunc(const float& Z) const
    {
        return 1.0f / (1.0f + FMath::Exp(-Z));
    }

```

```
FORCEINLINE float ReLUFunc(const float& Z) const
{
    return FMath::Max<float>(Z, 0);
}

FORCEINLINE float TanHFunc(const float& Z) const
{
    return 2.0f * SigmoidFunc(2.0f * Z) - 1.0f;
}
};
```

Б.2 Текст файлу NeuralNetwork.h

```
#pragma once
#include "SRMatrix.h"
#include "SRNeuralNetwork.generated.h"

USTRUCT()
struct SYMBOLRECOGNIZERPLUGIN_API FSRNeuralNetwork
{
    GENERATED_BODY()

    UPROPERTY()
        uint32 InputNodes = 0;
    UPROPERTY()
        uint32 HiddenNodes = 0;
    UPROPERTY()
        uint32 OutputNodes = 0;
    UPROPERTY()
        float LearningRate = 0;
    UPROPERTY()
        FSRDMatrix wih = FSRDMatrix(0,0,0);
    UPROPERTY()
        FSRDMatrix who = FSRDMatrix(0, 0, 0);
    UPROPERTY()
        bool bIsTrained = false;

    FSRNeuralNetwork() {};
    FSRNeuralNetwork(uint32 InInputNodes, uint32 InHiddenNodes,
uint32 InOutputNodes, float InLearningRate);

    void Train(const TArray<float>& InputList, const
TArray<float>& OutputList);
    FSRDMatrix Query(const TArray<float>& InputList) const;

    /*
    * @ Neural Item to verify.
    * @ QueryData Data to test in selected network.
    * @ AnswerIdx Output class id (symbol) that QueryData should
refer to
    * @ AcceptableAnserSize if the result is less then return 0.
```

```

    * @ DeltaBestAnswers if delta between 2 best answers is lest
then return 0.
    * @ return 1 when answer found or 0 otherwise.
    */
    static int32 GetQueryResult(const FSRNeuralNetwork& Neural,
const TArray<float>& QueryData, int32 AnswerIdx, float
AcceptableAsnwerSize = 0.5, float DeltaBestAnswers = 0.97);
};

```

Б.3 Текст файлу NeuralNetwork.cpp

```

#include "SRNeuralNetwork.h"
#include "SymbolRecognizerPlugin.h"

FSRNeuralNetwork::FSRNeuralNetwork(uint32 InInputNodes,
uint32 InHiddenNodes, uint32 InOutputNodes, float InLearningRate)
{
    InputNodes = InInputNodes;
    HiddenNodes = InHiddenNodes;
    OutputNodes = InOutputNodes;
    LearningRate = InLearningRate;
    wih = FSRDMatrix(HiddenNodes, InputNodes,
FMMatrixOperationDelegate::CreateLambda([&]() {return
FMath::FRand() * 0.05f * 2 - 0.05f; }));
    who = FSRDMatrix(OutputNodes, HiddenNodes,
FMMatrixOperationDelegate::CreateLambda([&]() {return
FMath::FRand() * 0.05f * 2 - 0.05f; }));
}

void FSRNeuralNetwork::Train(const TArray<float>& InputList,
const TArray<float>& OutputList)
{
    FSRDMatrix Inputs = FSRDMatrix(InputNodes, 1, InputList);
    FSRDMatrix Targets = FSRDMatrix(OutputNodes, 1, OutputList);

    FString InputsStr = "Inputs: " + Inputs.ToString();
    FString TargetsStr = "Targets: " + Targets.ToString();

    //activation FP
    FSRDMatrix HiddenInputs = wih * Inputs;
    FSRDMatrix HiddenOutputs =
HiddenInputs.ActivationOperation(FSRDMatrix::Sigmoid);
    // FSRDMatrix HiddenOutputs =
HiddenInputs.ActivationOperation(FSRDMatrix::TanH);
    // FSRDMatrix HiddenOutputs =
HiddenInputs.ActivationOperation(FSRDMatrix::ReLU);

    FSRDMatrix FinalInputs = who * HiddenOutputs;
    FSRDMatrix FinalOutputs =
FinalInputs.ActivationOperation(FSRDMatrix::Sigmoid);
}

```

```

        // FSRDMatrix FinalOutputs =
FinalInputs.ActivationOperation(FSRDMatrix::TanH);
        // FSRDMatrix FinalOutputs =
FinalInputs.ActivationOperation(FSRDMatrix::ReLU);

        //errors BP
        FSRDMatrix OutputErrors = (Targets - FinalOutputs);
        FSRDMatrix HiddenErrors = who.GetTranspose() * OutputErrors;

        //gradient descent learn
        //update weights from hidden to outputs
        who +=
OutputErrors.CompWiseMultiply(FinalOutputs).CompWiseMultiply(1.0f
- FinalOutputs) * HiddenOutputs.GetTranspose() * LearningRate;
        // who += OutputErrors.CompWiseMultiply(1.0f -
FinalOutputs).CompWiseMultiply(1.0f + FinalOutputs) *
HiddenOutputs.GetTranspose() * LearningRate;
        // who += OutputErrors.CompWiseMultiplyReLU(FinalOutputs) *
HiddenOutputs.GetTranspose() * LearningRate;
        //update weights from inputs to hidden
        wih +=
HiddenErrors.CompWiseMultiply(HiddenOutputs).CompWiseMultiply(1.0f
- HiddenOutputs) * Inputs.GetTranspose() * LearningRate;
        // wih += HiddenErrors.CompWiseMultiply(1.0f -
HiddenOutputs).CompWiseMultiply(1.0f + HiddenOutputs) *
Inputs.GetTranspose() * LearningRate;
        // wih += HiddenErrors.CompWiseMultiplyReLU(HiddenOutputs) *
Inputs.GetTranspose() * LearningRate;

        bIsTrained = true;
    }

    FSRDMatrix FSRNeuralNetwork::Query(const TArray<float>&
InputList) const
    {
        FSRDMatrix Inputs = FSRDMatrix(InputNodes, 1, InputList);
        FSRDMatrix HiddenInputs = wih * Inputs;
        FSRDMatrix HiddenOutputs =
HiddenInputs.ActivationOperation(FSRDMatrix::Sigmoid);
        // FSRDMatrix HiddenOutputs =
HiddenInputs.ActivationOperation(FSRDMatrix::TanH);
        FSRDMatrix FinalInputs = who * HiddenOutputs;
        FSRDMatrix FinalOutputs =
FinalInputs.ActivationOperation(FSRDMatrix::Sigmoid);
        // FSRDMatrix FinalOutputs =
FinalInputs.ActivationOperation(FSRDMatrix::TanH);

        return FinalOutputs;
    }

```

```

    int32 FSRNeuralNetwork::GetQueryResult(const
FSRNeuralNetwork& Neural, const TArray<float>& QueryData, int32
AnswerIdx, float AcceptableAsnwerSize /*= 0.5*/, float
DeltaBestAnswers /*= 0.97*/)
    {
        FSRDMatrix Result = Neural.Query(QueryData);

        int32 bestAnswer = -1;
        float bestSize = 0;
        float previousSize = 0;

        for (int32 answerIdx = 0; answerIdx < Result.R.Num();
answerIdx++)
        {
            if (Result.R[answerIdx].C[0] > bestSize)
            {
                previousSize = bestSize;
                bestSize = Result.R[answerIdx].C[0];
                bestAnswer = answerIdx;
            }
        }

        float bestPreviousDelta = bestSize - previousSize;

        if (bestAnswer == AnswerIdx
            && bestPreviousDelta > DeltaBestAnswers
            && bestSize >= AcceptableAsnwerSize)
        {
            return 1;
        }

        return 0;
    }

```

Б.4 Текст файла NetworkTrainingAsyncTask.h

```

// Copyright 2019 Piotr Macharzewski. All Rights Reserved.
#pragma once
#include "Runtime/Core/Public/Async/AsyncWork.h"
#include "SRNetworkTrainingAsyncTask.generated.h"

DECLARE_DELEGATE(FTrainingTaskCompleteDelegate);
DECLARE_DELEGATE(FTrainingTaskStopDelegate);

struct FSRNeuralNetwork;

/*
 * Stores training data for single symbol e.g. Symbol 'A' has
5 images,
 * so we have Answer pointing to Symbol ID and nested array
Inputs holding 28x28 pixes floats for each image referring to this
symbol ('A')

```

```

*/
USTRUCT(NotBlueprintable)
struct SYMBOLRECOGNIZERPLUGINEDITOR_API FSRTrainingDataSet
{
    GENERATED_BODY()

    TArray<TArray<float>> Inputs; //<< whole data for single
symbol inheres e.g. [Tex01][Fixels28x28]
    int32 Outputs;
    int32 Answer;//symbol id
    TArray<float> ExpectedOutput;
    FSRTrainingDataSet() {};
    FSRTrainingDataSet(TArray<TArray<float>>& InInputs, int32
InOutputs, int32 InAnswer)
        : Inputs(InInputs)
        , Outputs(InOutputs)
        , Answer(InAnswer)
    {
        ExpectedOutput.Reserve(Outputs + 1);
        for (int32 I = 0; I < Outputs; ++I)
            ExpectedOutput.Emplace(0.01f);

        //so its like:
        //A -> [0] = 0.01
        //B -> [1] = 0.01
        //C -> [Answer] = 0.99
        //D -> [3] = 0.01
        ExpectedOutput[InAnswer] = 0.99f;
    }

};

class SYMBOLRECOGNIZERPLUGINEDITOR_API
NetworkTrainingAsyncTask : public FNonAbandonableTask
{
    friend class USRToolManager;

    FSRNeuralNetwork& NeuralItem;
    UPROPERTY()
    TArray<FSRTrainingDataSet> TrainingsSet;
    int32 Outputs;
    int32 AllImagesCount;
    uint32 Inputs;
    uint32 Hidden;
    float Lr;
    int32 EpochsLimit;// if <= 0 then autotraining until
Accuracy is reached
    float AcceptableAccuracy;
    float DeltaBestAnswers;
    FTrainingTaskCompleteDelegate TrainingTaskComplete;
    FTrainingTaskStopDelegate TrainingTaskStop;
    static FThreadSafeBool bShouldStopSymbolTraining;
    static float Progress;

```

```

static int32 CurrentEpochs;
static TArray<float> SymbolsScores;

public:

NetworkTrainingAsyncTask(
    FSRNeuralNetwork& InNeuralItem
    , TArray<FSRTrainingDataSet>& InTrainigsSet
    , int32 InSymbolsCount
    , int32 InAllImagesCount
    , uint32 InInputs
    , uint32 InHidden
    , float InLr
    , int32 InEpochsLimit
    , float InAcceptableAccuracy, float InDeltaBestAnswers
    , FTrainingTaskCompleteDelegate InTrainingTaskComplete
    , FTrainingTaskStopDelegate InTrainingTaskStop
);

FORCEINLINE TStatId GetStatId() const
{
    RETURN_QUICK_DECLARE_CYCLE_STAT(NetworkTrainingAsyncTask,
    STATGROUP_ThreadPoolAsyncTasks);
}

void DoWork();
};

```

Б.5 Текст файла NetworkTrainingAsyncTask.cpp

```

#include "SRNetworkTrainingAsyncTask.h"
#include "SRNeuralNetwork.h"

FThreadSafeBool
NetworkTrainingAsyncTask::bShouldStopSymbolTraining = false;
float NetworkTrainingAsyncTask::Progress = 0.0f;
int32 NetworkTrainingAsyncTask::CurrentEpochs = 0;
TArray<float> NetworkTrainingAsyncTask::SymbolsScores = {};

NetworkTrainingAsyncTask::NetworkTrainingAsyncTask(FSRNeuralN
etwork& InNeuralItem, TArray<FSRTrainingDataSet>& InTrainigsSet,
int32 InSymbolsCount, int32 InAllImagesCount, uint32 InInputs,
uint32 InHidden, float InLr, int32 InEpochsLimit, float
InAcceptableAccuracy, float InDeltaBestAnswers,
FTrainingTaskCompleteDelegate InTrainingTaskComplete,
FTrainingTaskStopDelegate InTrainingTaskStop)
    : NeuralItem(InNeuralItem)
    , TrainigsSet(InTrainigsSet)
    , Outputs(InSymbolsCount)
    , AllImagesCount(InAllImagesCount)
    , Inputs(InInputs)

```

```

, Hidden(InHidden)
, Lr(InLr)
, EpochsLimit(InEpochsLimit)
, AcceptableAccuracy(InAcceptableAccuracy)
, DeltaBestAnswers(InDeltaBestAnswers)
, TrainingTaskComplete(InTrainingTaskComplete)
, TrainingTaskStop(InTrainingTaskStop)
{
Progress = 0.0f;
bShouldStopSymbolTraining = false;
CurrentEpochs = 0;
SymbolsScores.Reset();
SymbolsScores.Reserve(InSymbolsCount + 1);
for (int32 I = 0; I < InSymbolsCount; ++I)
    SymbolsScores.Emplace(0);
}

FCriticalSection Mutex;

void NetworkTrainingAsyncTask::DoWork()
{
int32 Epochs = 0;
bool bAutoTraining = EpochsLimit <= 0;
if (bAutoTraining)
{
    EpochsLimit = 3;
}

while (Epochs < EpochsLimit)
{
    Epochs++;
    CurrentEpochs = Epochs;

    if (bShouldStopSymbolTraining)
    {
        TrainingTaskStop.ExecuteIfBound();
        GLog->Log("-----");
        GLog->Log("TASK  BROKEN");
        GLog->Log("-----");
        return;
    }

    for (FSRTrainingDataSet& trainingSet : TrainigsSet)
    {
        for (TArray<float>& trainingData : trainingSet.Inputs)
        {
            if (bShouldStopSymbolTraining)
            {
                TrainingTaskStop.ExecuteIfBound();

```

```

        GLog->Log ("-----");
        -----");
        GLog->Log ("TASK BROKEN");
        GLog->Log ("-----");
        -----");
        return;
    }
    //...

    if (NeuralItem.bIsTrained == false)//initialize all
    necessary params if it was not in training before.
        NeuralItem = FSRNeuralNetwork(Inputs, Hidden,
    Outputs, Lr);

        NeuralItem.Train(trainingData,
    trainingSet.ExpectedOutput);

    }
}

//calculate training progress by comparing good answers
ratio.
float GoodAnswersCount = 0;
for (int32 SymbolIdx = 0; SymbolIdx < TrainigsSet.Num();
++SymbolIdx)
    //for (FSRTrainingDataSet& trainingSet : TrainigsSet)
    {
        int32 GoodAnswersPerSymbol = 0;
        for (TArray<float>& trainingData :
    TrainigsSet[SymbolIdx].Inputs)
        {
            GoodAnswersPerSymbol +=
    FSRNeuralNetwork::GetQueryResult(NeuralItem, trainingData,
    TrainigsSet[SymbolIdx].Answer, AcceptableAccuracy,
    DeltaBestAnswers);
        }

        SymbolsScores[SymbolIdx] = GoodAnswersPerSymbol /
    (float)TrainigsSet[SymbolIdx].Inputs.Num();
        GoodAnswersCount += GoodAnswersPerSymbol;
    }

    float Accuracy = GoodAnswersCount / AllImagesCount;
    FString AccuracyStr = FString::SanitizeFloat(Accuracy) + "
-- epochs: " + FString::FromInt(Epochs);

    GLog->Log("Accuracy: " + AccuracyStr + "
AcceptableAccuracy: " +
    FString::SanitizeFloat(AcceptableAccuracy));

    if (bAutoTraining)

```

```

{
    //Progress is static var.
    Progress = (Accuracy / AcceptableAccuracy);

    if (Accuracy < AcceptableAccuracy)
    {
        EpochsLimit += 1;
    }
}
else
{
    //Progress is static var.
    Progress = (Epochs / (float)EpochsLimit);
    //GLog->Log("Progress: " + FString::FromInt(Epochs) + "/"
+ FString::FromInt(EpochsLimit));
}
}

GLog->Log("-----
-----");
GLog->Log("End of NetworkTrainingAsyncTask calculation on
background thread");
GLog->Log("-----
-----");

TrainingTaskComplete.ExecuteIfBound();
}

```

Б.6 Текст файла Plugin.h

```

#pragma once

#include "SRNeuralNetwork.h"
#include "SRCanvasHandler.h"
#include "SymbolRecognizer.generated.h"

USTRUCT(NotBlueprintable)
struct FSRDrawLayerWrapper
{
    GENERATED_BODY()

    UPROPERTY()
    TArray<FSRDrawingLayer> DrawLayers;

    FSRDrawLayerWrapper() {}
    FSRDrawLayerWrapper(const TArray<FSRDrawingLayer>&
InDrawLayers) : DrawLayers(InDrawLayers) {}
};

UCLASS()

```

```

        class SYMBOLRECOGNIZERPLUGIN_API USymbolRecognizerData :
public UObject
    {
        GENERATED_BODY()

        friend class USymbolRecognizer;

        UPROPERTY()
        FString CurrentProfile = "TEST";
        UPROPERTY()
        TMap<FString, FSRNeuralNetwork> NeuralProfiles;
        UPROPERTY()
        TMap<FString, FSRDrawLayerWrapper> DrawLayers;

        bool ReadDrawLayers(const FString InProfile,
TArray<FSRDrawingLayer>& OutDrawLayers)
        {
            if (FSRDrawLayerWrapper* dlw = DrawLayers.Find(InProfile))
            {
                if (dlw->DrawLayers.Num() > 0)
                {
                    OutDrawLayers = dlw->DrawLayers;
                    return true;
                }
            }

            return false;
        }

        void WriteDrawLayers(const FString InProfile, const
TArray<FSRDrawingLayer>& InDrawLayers)
        {
            DrawLayers.Add(InProfile,
FSRDrawLayerWrapper(InDrawLayers));
        }
    };

    /**
     *
     */
    UCLASS(Blueprintable)
    class SYMBOLRECOGNIZERPLUGIN_API USymbolRecognizer : public
UObject
    {
        GENERATED_BODY()

        friend class USRCanvasHandler;

        public:
        static const FString SymbolRecognizerMountPoint;
        static const FString SRDataDir;

        static bool bFlipYAxis_Android;

```

```

    UFUNCTION(BlueprintCallable, meta = (WorldContext =
"WorldContextObject"), Category = "SymbolRecognizerPlugin")
        static USymbolRecognizer* CreateSymbolRecognizer(UObject*
WorldContextObject);
    UFUNCTION(BlueprintPure, Category =
"SymbolRecognizerPlugin")
        static USymbolRecognizer* GetSymbolRecognizerInstance();

    UFUNCTION(BlueprintCallable, Category =
"SymbolRecognizerPlugin")
        static void SetFlipYAxisForAndroid(bool bShouldFlip);

    UFUNCTION(BlueprintCallable, Category =
"SymbolRecognizerPlugin")
        void ShowAccuracyTestingWidget();
    UFUNCTION(BlueprintCallable, Category =
"SymbolRecognizerPlugin")
        void HideAccuracyTestingWidget();
    UFUNCTION(BlueprintCallable, Category =
"SymbolRecognizerPlugin")
        void SRShowPreviewCanvasWidget();
    UFUNCTION(BlueprintCallable, Category =
"SymbolRecognizerPlugin")
        void SRHidePreviewCanvasWidget();

    /*
    * Find out whether currently drawn pattern is acceptable.
    * @param ExpectedSymbol          index of a symbol we want to
test.
    * @param AccuracyTreshold        (0-1) drawing accuracy must
be bigger to pass.
    * @param CompareToMostAccurateSymbol return false if found
more accurate symbol than ExpectedSymbol.
    * @return true when drawn pattern is accurate enough.
    */
    UFUNCTION(BlueprintCallable, Category =
"SymbolRecognizerPlugin")
        bool TestDrawing(int32 ExpectedSymbol = 0, float
AccuracyTreshold = 0.3f, bool bCompareToMostAccurateSymbol = true)
const;

    /*
    * Test current drawing and return the most accurate
SymbolId.
    * @param AccuracyTreshold        (0-1) drawing accuracy must be
bigger to pass.
    * @return SymbolId if found, otherwise -1.
    */
    UFUNCTION(BlueprintCallable, Category =
"SymbolRecognizerPlugin")
        int32 GetMostAccurateSymbol(float AccuracyTreshold = 0.3f)
const;

```

```

TArray<float> GetAccuracyList() const;

UFUNCTION(BlueprintCallable, Category =
"SymbolRecognizerPlugin")
    class USRCanvasHandler* GetCanvasHandler() const;
    FORCEINLINE int32 GetSymbolTextureSize() const { return
NeuralTextureSize; };

    bool AssignDrawingLayers(TArray<FSRDrawingLayer>&
OutDrawingLayers);

    /*
    * Adds new drawing line. Should be called on drawing button
pressed.
    */
    UFUNCTION(BlueprintCallable, Category =
"SymbolRecognizerPlugin")
        void BeginDrawing();
    /*
    * Extends drawing line. Should be called in Tick.
    */
    UFUNCTION(BlueprintCallable, Category =
"SymbolRecognizerPlugin")
        void Draw(FVector2D InDrawPosition);
    /*
    * Breaks drawing line. Should be called on drawing button
release.
    */
    UFUNCTION(BlueprintCallable, Category =
"SymbolRecognizerPlugin")
        void EndDrawing();
    /*
    * Clears current drawing, so methods TestDrawing and
GetMostAccurateSymbol should be called before.
    */
    UFUNCTION(BlueprintCallable, Category =
"SymbolRecognizerPlugin")
        void ClearCanvas();

    FSRNeuralNetwork& GetNeuralNetworkRef(bool bTryLoad = true);

    void SaveNeuralProfile(const FString InProfile,
FSRNeuralNetwork& InNeuralNetwork);
    void SaveNeuralProfile(const FString InProfile);
    bool LoadNeuralNetworkFromSRData(FSRNeuralNetwork&
NeuralData);
    bool SelectProfile(const FString InProfile, bool
bAddEmptyIfNotFound = false, bool bShouldSave = true);

    void SaveSRData();
    void LoadSRData();
    bool DeleteProfile(const FString InProfile);

```

```

        UFUNCTION(BlueprintCallable, Category =
"SymbolRecognizerPlugin")
        FORCEINLINE bool CheckIsDrawing() const { return bIsDrawing;
}

        /*
        * Switch profile to load different NeuralData file.
        */
        UFUNCTION(BlueprintCallable, Category =
"SymbolRecognizerPlugin")
        void ChangeProfile(FString InProfile = "TEST", bool
bShouldSave = true);

        FString GetCurrentProfile() const;
        TArray<FString> GetProfilesNames() const;

protected:
        virtual void PostLoad() override;

private:
        UPROPERTY(Transient)
        mutable class USRCanvasHandler* NeuralTexture = nullptr;
        UPROPERTY()
        mutable USymbolRecognizerData* SRData = nullptr;
        UPROPERTY()
        UPackage* SRDataPackage = nullptr;

        UPROPERTY(Transient)
        FSRNeuralNetwork NeuralNetwork;
        bool bIsLoaded = false;
        UPROPERTY(EditAnywhere, Category = "Training")
        int32 NeuralTextureSize = 28;
        bool bIsDrawing = false;
        bool bAddNewDrawSpot = false;

        FORCEINLINE FString GetSRDataPackageName() const;

};

```

ДОДАТОК В
Слайди презентації

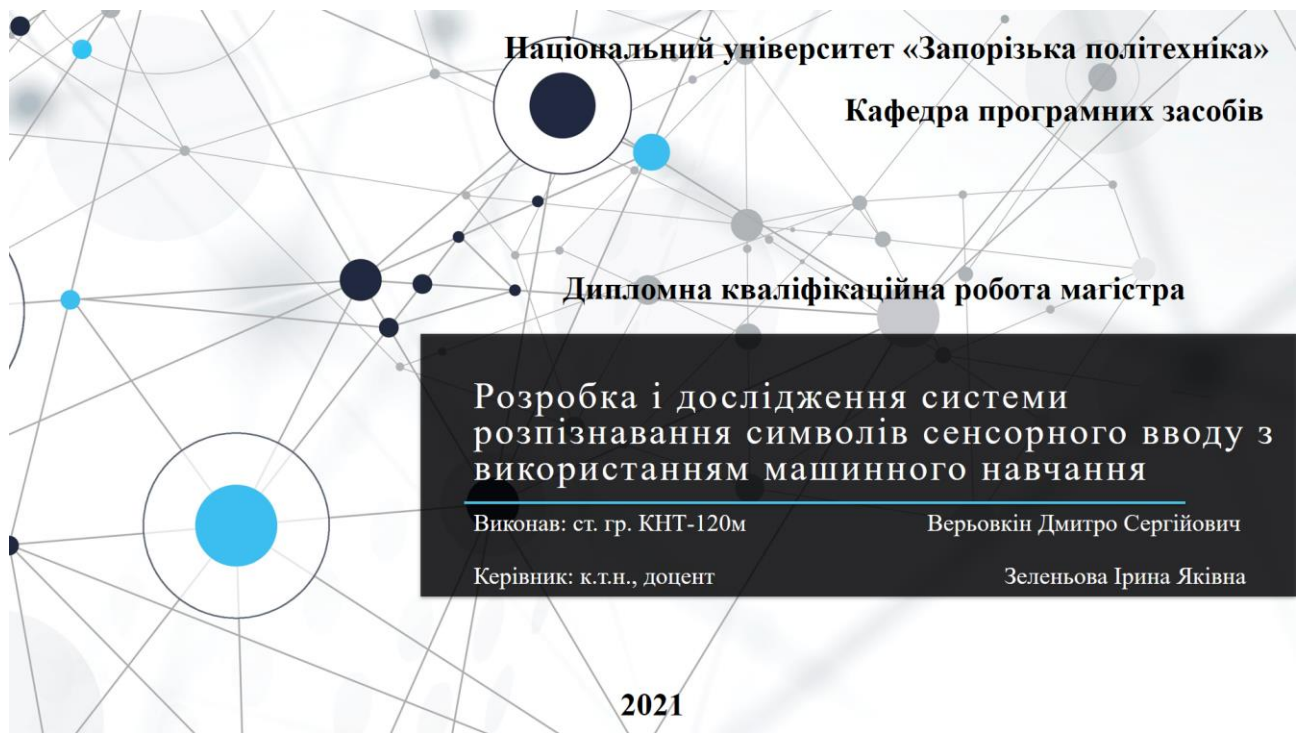


Рисунок В.1 – Слайд №1

Об'єкт, предмет і мета роботи

- Об'єкт дослідження – програмні засоби для розпізнавання символів.
- Предмет дослідження – нейронні мережі для розпізнавання рукописних символів.
- Мета роботи – підвищення рівня автоматизації, точності та швидкості навчання процесу розпізнавання рукописних символів шляхом створення програмної системи на основі методів і моделей штучного інтелекту.

Рисунок В.2 – Слайд №2

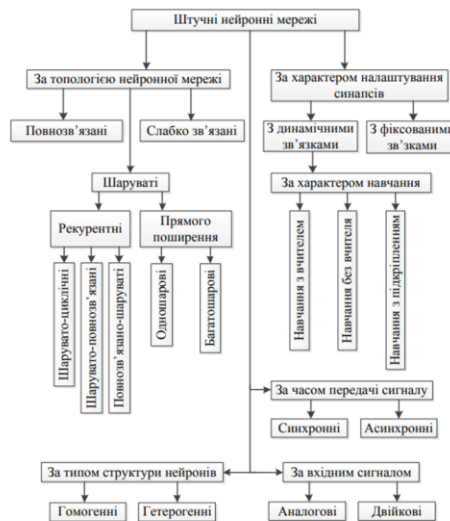
Завдання

Дослідити та реалізувати НМ на основі багатошарового персептрону для автоматизації та спрощення процесу навчання для розпізнавання символів. Додатково розглянути функції активації нейронів прихованого шару та їх вплив на поведінку НМ. Дослідити вплив методу розрахунку витрат на швидкість навчання та точність розпізнавання НМ.

3

Рисунок В.3 – Слайд №3

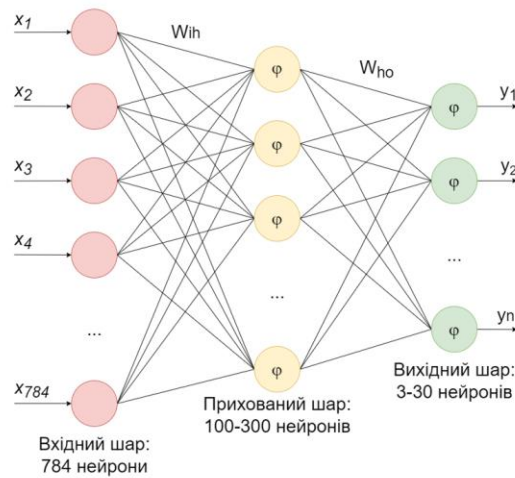
Класифікація нейронних мереж



4

Рисунок В.4 – Слайд №4

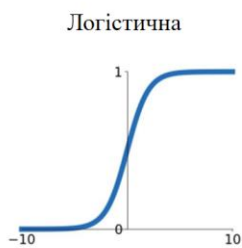
Багатошаровий персептрон з одним прихованим шаром



5

Рисунок В.5 – Слайд №5

Функції активації



$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$$



$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$\frac{\partial \tanh(x)}{\partial x} = (1 + \tanh(x))(1 - \tanh(x))$$



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

$$\frac{\partial \text{ReLU}(x)}{\partial x} = \begin{cases} 0 & (x < 0) \\ 1 & (x > 0) \end{cases}$$

6

Рисунок В.6 – Слайд №6

Цільова функція

Середньоквадратична

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

$$\frac{\text{MSE}}{\partial \hat{y}} = \frac{2}{n} \sum_{i=1}^n (y_i - \hat{y}_i)$$

Перехресної ентропії

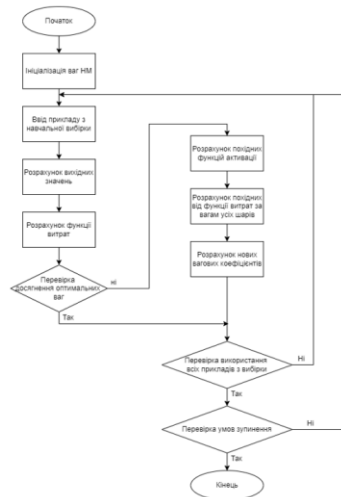
$$\text{CE} = - \sum_{i=1}^C y_i \log \hat{y}_i$$

$$\frac{\text{CE}}{\partial \hat{y}} = \hat{y} - y \quad \text{*(при використанні Softmax в вихідному шарі)}$$

7

Рисунок В.7 – Слайд №7

Алгоритм зворотнього розповсюдження похибки



8

Рисунок В.8 – Слайд №8

Порівняння мов програмування

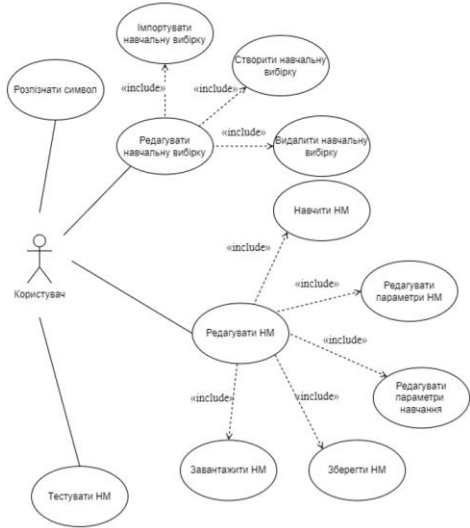
Критерії порівняння	Мова програмування		
	C++	C#	Python
Кількість сторонніх бібліотек	±	±	+
Типізація	Статична	Статична	Динамічна
Швидкодія	Висока	Середня	Низька (Середня*)
Складність вивчення	Висока	Середня	Низька
Швидкість розробки	Середня	Висока	Висока
Збирання сміття	–	+	+
Метод виконання коду	Компіляція у машинний код	Компіляція у байт-код	Інтерпретація

*(при використанні сторонніх бібліотек на C/C++)

9

Рисунок В.9 – Слайд №9

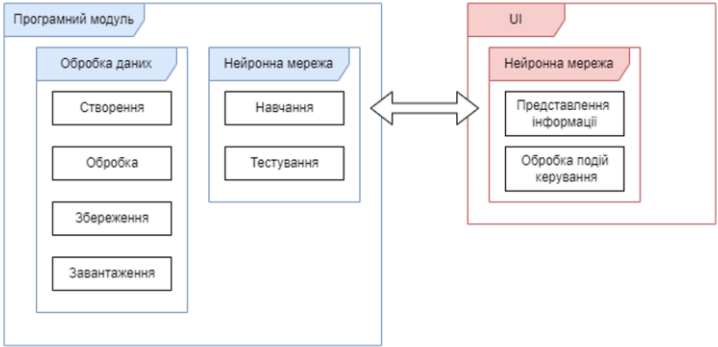
Діаграма прецедентів



10

Рисунок В.10 – Слайд №10

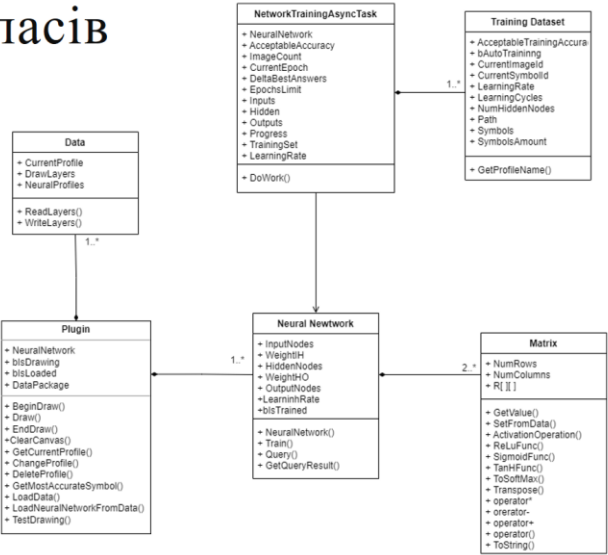
Модульне представлення ПЗ



11

Рисунок В.11 – Слайд №11

Діаграма класів



12

Рисунок В.12 – Слайд №12

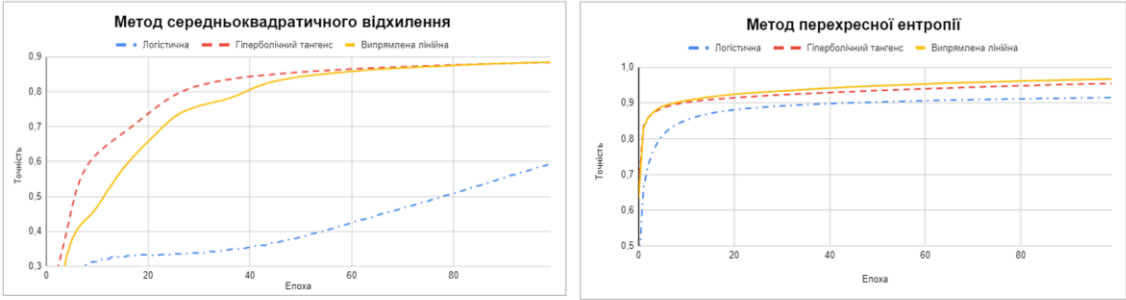
Результати експерименту

Метод визначення витрат:	Метод середньоквадратичного відхилення		Метод перехресної ентропії	
Функція активації:	Витрати	Точність	Витрати	Точність
Логістична	0,0645	60,65%	0,2945	91,89%
Гіперболічний тангенс	0,0195	89,24%	0,163	95,25%
Випрямлена лінійна	0,0197	89,42%	0,1186	96,30%

13

Рисунок В.13 – Слайд №13

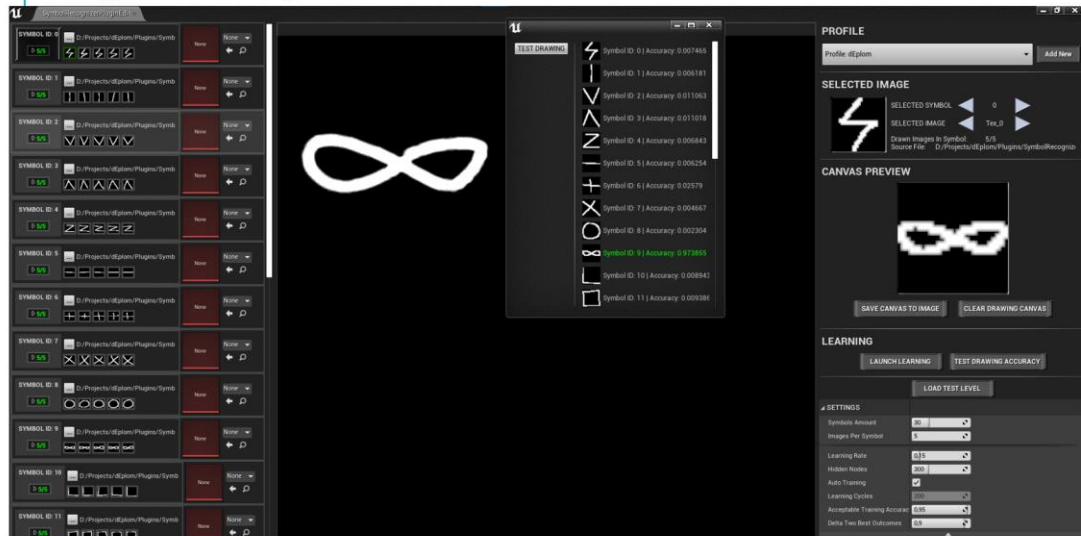
Результати експерименту



14

Рисунок В.14 – Слайд №14

Приклад роботи з ПЗ



15

Рисунок В.15 – Слайд №15

Висновки

- Проведено аналіз предметної області, у результаті якого було обрано архітектуру НМ. У результаті огляду існуючих програмних засобів була визначена недоцільність їх використання у даній роботі. Було визначено функціональні вимоги до програми.
- Розглянуто метод зворотного розповсюдження похибки для навчання НМ. Також розглянуто функції активації нейронів та методи визначення витрат.
- Проведено проектування архітектури ПЗ для реалізації. Було виконано порівняння засобів реалізації ПЗ, в результаті чого було обрано ігровий рушій UE4, мову програмування C++ та середовище розробки Visual Studio.
- Проведено експериментальну оцінку ефективності методів визначення витрат і функцій активації нейронів прихованого шару. Метод перехресної ентропії разом з випрямленою лінійною функцією активації показав найкращі результати при задачі багатокласової класифікації, тому в ПЗ реалізовано саме цей варіант.
- Створено програмний застосунок, який у формі плагіна підключається до ігрового рушія Unreal Engine 4 і вирішує задачу створення та навчання нейронної мережі. Розроблено програмну систему для розпізнавання рукописних символів, яка автоматизує та прискорює процес створення та навчання нейронної мережі.
- Галузь використання – розробка ігор для комп'ютерних і мобільних платформ з символьним керуванням.

16

Рисунок В.16 – Слайд №16