Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інтелектуальних інформаційних технологій та автоматизації
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Інформаційна технологія оптичного розпізнавання тексту»

Виконав: студент 2-го курсу, групи 2КН-22м
спеціальності 122 «Комп'ютерні науки»
(шифр і назва напряму підготовки, спеціальності)
Горбатюк О.О.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. КН
Барабан С.В.
(прізвище та ініціали)
«07» 12 2023 р.

Опонент: к.т.н., доц. каф. АІІТ
Варчук І.В.
(прізвище та ініціали)
«07» 12 2023 р.

Допущено до захисту
Завідувач кафедри КН
д.т.н., проф. Яровий А.А.
(прізвище та ініціали)
«09. 12» 2023 р.

Вінниця ВНТУ - 2023 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма – «Системи штучного інтелекту»

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
д.т.н., проф. Яровий А.А.
_____
(підпис)
"29" 08 2023 року

З А В Д А Н Н Я
на магістрекську кваліфікаційну роботу
студенту
Горбатюку Олександру Олександровичу

1 Тема роботи: «Інформаційна технологія оптичного розпізнавання тексту».

Керівник роботи: Барабан Сергій Володимирович, к.т.н., доцент.

затверджені наказом вищого навчального закладу «18» 09 2023 року № 247

2 Строк подання студентом роботи 13.11.2023

3 Вихідні дані до роботи: 1) кількість навчальних образів – не менше 26од; 2) кількість шрифтів для розпізнавання – не менше 5шт; 3) види шрифтів – стандартні та курсивні; 4) Мова програмування – об'єктно-орієнтована.

4 Зміст пояснювальної записки (перелік питань, які потрібно розробити): вступ, аналіз предметної області розпізнавання тексту; розробка інформаційної технології оптичного розпізнавання тексту; програмна реалізація інформаційної технології оптичного розпізнавання тексту; економічна частина, висновки, перелік використаних джерел, додатки.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): схема алгоритму функціонування інформаційної технології оптичного розпізнавання тексту; діаграма класів програмного забезпечення

оптичного розпізнавання тексту; головні вінка розробленого програмного забезпечення.

6 Консультанти розділів проекту (роботи)

Консультанти розділів роботи в таблиці 1.

Таблиця 1 - Консультанти роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Барабан С.В., к.т.н., доцент каф. КН	29.08.23	10.11.23
4	Ратушняк О. Г., к.т.н, доц. каф. ЕПВМ	5.11.23	30.11.23

7 Дата видачі завдання 29.08.2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області розпізнавання тексту	07.09.23	
2	Розробка інформаційної технології оптичного розпізнавання тексту	24.09.23	
3	Програмна реалізація інформаційної технології оптичного розпізнавання тексту	15.10.23	
4	тестування та аналіз результатів роботи розробленої програми	29.10.23	
5	Розробка інструкції користувача	05.11.23	
6	Оформлення матеріалів до захисту МКР	10.11.23	

Студент _____ Горбатюк О.О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Барабан С.В.
(підпис)

АНОТАЦІЯ

УДК 621.374.415

Горбатюк О.О. Інформаційна технологія оптичного розпізнавання тексту. Магістерська кваліфікаційна робота зі спеціальності 122 – комп'ютерні науки, освітня програма - Системи штучного інтелекту. Вінниця: ВНТУ, 2023. 103с.

На укр. мові. Бібліогр.: 20 назв; рис.: 20; табл. 14.

Дана магістерська кваліфікаційна робота присвячена розробці інформаційної технології оптичного розпізнавання тексту.

В роботі були розглянуті основні методи розпізнавання тексту, визначені їх переваги і недоліки, та на основі цього був зроблений вибір на користь нейронної мережі багатошаровий персептрон. Було досліджено структуру, математичну модель та алгоритм роботи обраної мережі, який слугував базисом для розробки алгоритму роботи програмного забезпечення. Було визначено оптимальне для потенційного користувача програмне середовище, алгоритм роботи програми та в кінцевому підсумку на основі проведених досліджень був побудований програмний засіб розпізнавання тексту. Проект реалізовано за допомогою мови програмування C#.

В розділі економічної частини проведено оцінювання комерційного потенціалу розробки інформаційної технології оптичного розпізнавання тексту, спрогнозовано витрати на виконання наукової роботи та впровадження результатів, які склали 121690,99 грн, розраховано період окупності – 1,8 року.

Ключові слова: інформаційна технологія, оптичне розпізнавання тексту, нейронна мережа.

ABSTRACT

Horbatyuk O.O. Information technology of optical text recognition. Master's thesis on specialty 122 - computer science, educational program - Artificial intelligence systems. Vinnytsia: VNTU, 2023. 103p.

In Ukrainian speech Bibliography: 20 titles; Fig.: 20; table 14.

This master's thesis is devoted to the development of information technology for optical text recognition.

In the work, the main methods of text recognition were considered, their advantages and disadvantages were determined, and based on this, a choice was made in favor of a multi-layer perceptron neural network. The structure, mathematical model, and operation algorithm of the selected network were investigated, which served as the basis for the development of the software operation algorithm. The software environment optimal for the potential user, the algorithm of the program was determined, and finally, based on the conducted research, a software tool for text recognition was built. The project was implemented using the C# programming language.

In the section of the economic part, an assessment of the commercial potential of the development of the information technology of optical text recognition was carried out, the costs for the implementation of the scientific work and the implementation of the results were predicted, which amounted to UAH 121,690.99, and the payback period was calculated - 1.8 years.

Keywords: information technology, optical text recognition, neural network.

ЗМІСТ

ВСТУП

Актуальність теми дослідження. В галузі штучного інтелекту однією з ключових практичних задач є розпізнавання тексту та символів, що представляє собою конкретний випадок загальної задачі класифікації об’єктів (сигналів, явищ, процесів) [1]. Ця задача широко поширена і вирішується людиною дуже часто.

Оптичне розпізнавання тексту - це електронний процес перетворення зображень рукописного, машинописного або друкованого тексту в послідовність кодів, які можуть бути використані для представлення в текстовому редакторі [2]. Цей метод широко використовується для конвертації книг і документів в електронний формат, автоматизації систем обліку в бізнесі та публікації тексту в Інтернеті. Оптичне розпізнавання тексту надає можливість редагувати текст, здійснювати пошук слова або фрази, зберігати його в компактній формі, демонструвати або роздруковувати матеріал, зберігаючи якість, аналізувати інформацію та застосовувати до тексту електронний переклад, форматування або перетворення в мовлення. Деякі системи оптичного розпізнавання тексту можуть відтворювати вихідне форматування тексту, включаючи зображення, колонки та інші нетекстові компоненти [1].

Оптичне розпізнавання тексту є об'єктом досліджень у галузях розпізнавання образів, штучного інтелекту та комп'ютерного зору. Основним недоліком програмних засобів розпізнавання символів є їхня невисока точність розпізнавання. Для вирішення більш складних завдань у сфері розпізнавання часто використовуються інтелектуальні системи, такі як штучні нейронні мережі. Нейронні мережі є частиною штучного інтелекту, де для обробки сигналів використовуються принципи, подібні тим, які мають місце в нейронах живих організмів.

У даній роботі нейронні мережі використовуються для вирішення задачі розпізнавання рукописних та друкованих символів з використанням багатошарового персептрона, що навчається за алгоритмом зворотного поширення помилки.

Зв'язок роботи з науковими програмами, планами, темами. Магістерська робота виконана відповідно до напрямку наукових досліджень кафедри

комп'ютерних наук Вінницького національного технічного університету 22 К1 «Моделі, методи, технології та пристрої інтелектуальних інформаційних систем управління, економіки, навчання та комунікацій» та плану наукової та навчально-методичної роботи кафедри.

Мета і завдання досліджень. Метою магістерської кваліфікаційної роботи є підвищення достовірності оптичного розпізнавання тексту програмними засобами за рахунок використання нейронної мережі.

Для досягнення мети розробки необхідно виконати такі задачі:

- провести аналіз проблеми розв'язання задачі розпізнавання тексту;
- проаналізувати існуючі методи вирішення задачі розпізнавання тексту. Обрати та обґрунтувати вибір методу, що задовольняє мету даного магістерського дослідження;
- удосконалити математичну модель розпізнавання тексту;
- сформулювати стадії інформаційної технології, розробити структуру та алгоритм роботи програмного засобу;
- виконати програмну реалізацію запропонованої інформаційної технології;
- провести тестування програмного продукту та виконати аналіз отриманих результатів;
- здійснити економічні розрахунки доцільності розробки нової інформаційної технології.

Об'єкт дослідження – процес розпізнавання тексту з використанням штучних нейронних мереж.

Предмет дослідження – програмні засоби розпізнавання тексту з використанням нейронної мережі.

Методи дослідження. У роботі використані наступні методи наукових досліджень: системного аналізу, розпізнавання образів, теорії штучних нейронних мереж для реалізації інформаційної технології розпізнавання тексту, методи математичної статистики для розробки процесу розпізнавання символів та обрахунків результатів експериментів із програмним засобом, об'єктно-орієнтованого програмування.

Наукова новизна одержаних результатів.

- Набула подальшого розвитку інформаційна технологія оптичного розпізнавання тексту, яка відрізняється використанням штучної нейронної мережі багатошаровий персептрон, що дозволило підвищити достовірність розпізнавання тексту.

Практичне значення одержаних результатів полягає в тому, що на основі проведених досліджень розроблено програмне забезпечення розпізнавання тексту.

Запропонована інформаційна технологія сприяє підвищенню достовірності процесу розпізнавання тексту, зокрема:

– розроблено алгоритм навчання нейронної мережі багатошаровий персептрон;

– розроблено алгоритм роботи програмного забезпечення розпізнавання тексту на основі штучної нейронної мережі багатошаровий персептрон;

– розроблено програмні засоби для розпізнавання тексту на основі штучної нейронної мережі багатошаровий персептрон.

Достовірність теоретичних положень магістерської кваліфікаційної роботи підтверджується коректністю постановки завдання, коректністю використання математичного апарату методів дослідження, експериментальними дослідженнями тестування програмної реалізації інформаційної технології розпізнавання символів. Адекватність розроблених математичних моделей підтверджується результатами експериментальних досліджень.

Особистий внесок здобувача. Усі результати, що наведені у магістерській кваліфікаційній роботі, отримані самостійно. У працях, які написано у співавторстві, здобувачу належать: розробка структури інформаційної технології оптичного розпізнавання тексту [1].

Апробація результатів роботи. Результати роботи були апробовані на конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2023)» (м. Вінниця, Україна, 2023 р.) [1].

Публікації. За результатами магістерської кваліфікаційної роботи опубліковано тезу доповіді на конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2023)» [1].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ РОЗПІЗНАВАННЯ ТЕКСТУ

1.1 Аналіз задачі розпізнавання тексту

На сьогоднішній день існує значна кількість потужних програм для розпізнавання символів. Однак важливо відзначити, що здатність людини читати текст низької якості значно перевершує можливості комп'ютера.

Кожен друкований текст має важливу характеристику - шрифти, які використовуються. Виходячи з цього, існують два основних класи алгоритмів розпізнавання друкованих символів: шрифтовий та безшрифтовий. Шрифтові або шрифтозалежні алгоритми використовують апріорну інформацію про конкретний шрифт літер. Це передбачає, що програмі надається повноцінний зразок тексту, написаного цим шрифтом. Програма вимірює та аналізує різні характеристики шрифту, записуючи їх у свою базу еталонних характеристик. Після цього програма готова до розпізнавання тексту, написаного цим конкретним шрифтом [3].

Однак існують недоліки цього підходу:

- алгоритм має заздалегідь знати шрифт, якого він очікує для розпізнавання, тобто він повинен мати в базі різні характеристики цього шрифту;

- якість розпізнавання тексту, написаного будь-яким шрифтом, залежить від того, наскільки добре характеристики цього шрифту корелюють із зразками шрифтів, які вже є в базі програми.

Ці фактори обмежують універсальність таких алгоритмів. Програма, що використовує шрифтовий метод розпізнавання символів, вимагає від користувача спеціальних знань щодо шрифтів загалом, їхніх груп та відмінностей один від одного, а також інформації про шрифти, які використовуються в конкретному документі користувача. У випадку, якщо паперовий документ не був створений самим користувачем, інформації про те, які саме шрифти використовуються у цьому документі, може бути важко визначити.

Натомість, у випадку шрифтового підходу існує перевага, яку активно використовують на сьогодні і, очевидно, будуть використовувати у майбутньому. Зокрема, завдяки детальній апріорній інформації про символи можна створювати

досить точні та надійні алгоритми розпізнавання. Загалом, у процесі розробки шрифтового алгоритму, на відміну від безшрифтового, надійність розпізнавання символу стає інтуїтивно зрозумілою та математично вираженою величиною. Ця величина визначається як відстань у метричному просторі між еталонним символом, який надається програмі під час навчання, та символом, який програма намагається розпізнати.

Другий клас алгоритмів - це безшрифтові або шрифтонезалежні алгоритми, які не мають апріорних знань про символи, що подаються на вхід. Ці алгоритми вимірюють та аналізують різні характеристики (ознаки), що властиві буквам незалежно від шрифту та абсолютного розміру, якими вони надруковані. У випадку безшрифтового алгоритму, процес навчання може бути відсутнім, іноді характеристики символів вимірює, кодує та вводить в базу програми сам користувач. Однак на практиці рідко виникають ситуації, коли такий підхід повністю вирішує поставлену задачу. Загалом, більш загальний спосіб створення бази характеристик полягає в навчанні програми за допомогою реальних прикладів символів.

Недоліком цього підходу є менша якість розпізнавання у порівнянні з шрифтовими алгоритмами. Це пов'язано з тим, що рівень узагальнення при вимірах характеристик символів набагато вищий, ніж у випадку шрифтозалежних алгоритмів. Фактично, це означає, що різні допуски та спрощення при вимірах характеристик символів для роботи безшрифтових алгоритмів можуть бути в 2-20 разів більші, ніж у випадку шрифтових.

Переваги цього підходу тісно пов'язані з його недоліками. Основні переваги включають:

- універсальність -з одного боку, цей підхід може бути застосований у випадках значної різноманітності символів, які можуть надійти на вхід системи. З іншого боку, завдяки здатності узагальнювати, такі алгоритми можуть використовувати накопичені знання за межі навчальної вибірки, розпізнавати символи, які суттєво відрізняються від тих, які були включені в навчальний набір;

- технологічність - процес навчання безшрифтових алгоритмів є простішим і більш інтегрованим, оскільки навчальна вибірка не розбивається на різні класи. Це

також означає відсутність потреби підтримувати в базі характеристик різні умови спільного існування цих класів. Технологічність виявляється і в тому, що часто можна автоматизувати процедури навчання;

- зручність у використанні програми - коли програма ґрунтується на безшрифтових алгоритмах, користувачеві не потрібно мати певні знання про сторінку, яку він вводить в пам'ять комп'ютера. Це спрощує інтерфейс користувача програми за рахунок відсутності потреби в різноманітних опціях та діалогах для навчання та управління базою характеристик. У такому випадку процес розпізнавання може бути представлений користувачеві як "чорний ящик", де користувач повністю не контролює хід процесу розпізнавання. Це призводить до розширення кола потенційних користувачів, оскільки включає людей, які мають мінімальні навички комп'ютерного користування.

Програма, яка розробляється, повинна виконувати такі функції:

- виконувати розпізнавання друкованих і намальованих вручну літер англійського алфавіту;
- кількість навчальних образів, за допомогою яких ми будемо навчати нейронну мережу, повинна бути не менше 26 (одна літера – один образ);
- кількість шрифтів для розпізнавання повинна бути не менше 5. Для нашої програми виберемо такі шрифти: Arial, Courier, Tahoma, Times New Roman, Verdana;
- розпізнавати як і стандартний, так і курсивний тип шрифтів, наведених вище;
- в програмі має бути область для малювання символу, який програма буде розпізнавати. Розмір області – 150x150 пікселів;
- в програмі має бути можливість вибору кількості шарів нейронної мережі;
- в програмі має бути можливість налаштування швидкості навчання нейронної мережі;
- в програмі має бути можливість вибору межі похибки при навчанні нейронної мережі;
- програма має виводити інформацію про сумарну похибку після навчання мережі;

- програма має виводити інформацію про кількість невірно класифікованих образів після навчання мережі;

- програма має наочно показувати процес навчання нейронної мережі: поточний час, який триває навчання мережі, поточну похибку навчання тощо;

Програма також повинна володіти легким та приємним для користувача графічним інтерфейсом.

1.2 Аналіз методів розпізнавання тексту

Існує ряд методів для розпізнавання тексту, і їхні відмінності головним чином полягають у виборі характеристик для розпізнавання. Ці методи можна класифікувати у чотири загальні підходи до розпізнавання образів:

- структурні методи;
- статистичні методи;
- співставлення шаблонів;
- нейронні мережі.

Важливо зауважити, що ці підходи не завжди є абсолютно незалежними або відокремленими один від одного, іноді метод, застосований в одному підході, може також відповідати іншому.

Операції співставлення шаблонів визначають ступінь подібності між двома векторами в просторі ознак, які можуть представляти групи пікселів, фігур, вигини тощо. Такі методи можна розділити на три класи: пряме співставлення, пружне співставлення і релаксаційне співставлення.

Статистичні методи використовують статистичні функції для прийняття рішень і оптимальні критерії, які визначають ймовірність того, що спостережуване зображення належить до певного класу.

Низка визнаних методів розпізнавання символів входить у цей ряд:

- правило k-найближчих сусідів (k-NN). Цей популярний непараметричний метод розпізнавання визначає апостеріорну ймовірність невідомого шаблону приблизно через частоту його найближчих сусідів. Незважаючи на те, що цей підхід

демонструє досить високі результати, важливо відзначити, що для його класифікації потрібні значні обчислювальні потужності;

- прихована модель Маркова (ПММ). Цей метод вирішення задачі розпізнавання є ефективним інструментом. ПММ – це двічі стохастичний процес, який включає неспостережуваний процес (відсільний як "прихований"), який може спостерігатися через інший випадковий процес, що генерує послідовність спостережень. Ці ймовірнісні моделі є потужним засобом моделювання мови та інших реальних сигналів. Вони володіють багатьма бажаними властивостями для ефективного моделювання символів чи слів, зокрема – можливістю автоматичного навчання без необхідності попереднього маркування сегментованих даних. ПММ широко використовуються для розпізнавання рукописних слів, часто в поєднанні з іншими підходами, такими як стохастичні граматики і нейронні мережі. Виділяють два основні підходи в рамках цього методу: модель-дискримінантні ПММ і шлях-дискримінантні ПММ. У першому випадку модель будується для кожного класу (слів, літер або сегментів) під час підготовки, в другому ж одна ПММ створюється для усієї мови або контексту. Загалом продуктивність обох підходів порівнюється, але вони видаються важливими для ефективного розпізнавання.

Ці два методи представляють собою лише частину широкого спектру підходів до розпізнавання символів, в якому кожен метод має свої особливості та переваги в різних умовах та завданнях.

- метод опорних векторів (SVM). Цей метод, оснований на статистичній теорії навчання та методах квадратичної оптимізації, став досить впливовим у сфері машинного навчання. SVM є в основному двійковим класифікатором, але може бути розширений, об'єднуючи кілька SVM для створення системи класифікації для декількох класів. Останні роки характеризуються зростанням популярності SVM в машинному навчанні завдяки його видатній узагальнюючій здатності.

Своєрідність SVM полягає в тому, що він ефективно працює в умовах складних класифікаційних завдань. Він старанно визначає границі між класами, спираючись на вектори, що найкраще розділяють класи, тобто опорні вектори. Цей метод працює

добре як у лінійно роздільних, так і в нелінійно роздільних просторах, завдяки використанню ядерної трансформації.

Важливою перевагою SVM є його здатність до узагальнення та ефективності на нових, раніше не бачених даних. Він також відзначається високою точністю в порівнянні з іншими методами, зокрема в умовах обмежених ресурсів та високих вимог до швидкості обчислень.

Цей підхід виявився особливо корисним в задачах класифікації образів та векторів даних у великих наборах даних. Його ефективність у вирішенні складних завдань забезпечила йому статус потужного інструменту у сфері машинного навчання.

Інші статистичні підходи, такі як байєсівські або поліноміальні дискримінантні класифікатори, також існують, але вони менше популярні для розглядуваної задачі і, в більшості випадків, виявляються менш ефективними, подаючи гірші результати. Щодо структурних методів, символи у цих підходах представлені як об'єднання структурних символів-примітивів, отриманих з почерку, які піддаються кількісній оцінці. Також, можна визначити два основних класи структурних методів: граматичні та графічні [4].

Найбільш універсальним методом вирішення даної задачі є використання нейромереж. Нейронна мережа представляє собою обчислювальну структуру, що складається з штучних нейронів, які є абстракцією нервових клітин людини. Створені з метою імітації людського мозку, ці структури широко використовуються для розпізнавання образів, обробки даних та вирішення задач апроксимації функцій. Головні переваги нейронних мереж полягають в їхній здатності автоматичного навчання на основі вибірок, ефективності на зашумлених даних, можливості паралельної реалізації та використанні в обробці великих баз даних. У розглядуваній області, нейронні мережі широко використовуються та досягають перспективних результатів, зокрема в розпізнаванні рукописних символів. Цьому підходу притаманні різноманітні методи, серед яких варто виокремити нечіткі нейронні мережі, мережу Хеммінга, мережу Хопфілда, самоорганізуючі карти Кохонена та інші [5].

1.3 Вибір і обґрунтування аналогу до програмного забезпечення розпізнавання тексту

Програми розпізнавання тексту, також відомі як OCR (Optical Character Recognition), є інноваційними інструментами, спрямованими на автоматизацію процесу перетворення тексту, написаного або друкуваного на фізичних носіях, у цифровий формат. Ці програми використовуються в різноманітних галузях, включаючи бізнес, освіту, медицину, бібліотечну справу та багато інших.

Основні функції програм розпізнавання тексту включають:

- сканування та зчитування. Програми OCR можуть сканувати документи або зчитувати фотографії тексту, отримуючи зображення символів для подальшого аналізу;

- розпізнавання кодування. Вони визначають кодування символів, враховуючи шрифти, стилі та розміри, для точного перетворення на цифровий текст;

- перетворення на цифровий текст. OCR конвертує фізичний текст у цифровий, що дозволяє подальшу обробку та редагування за допомогою текстових редакторів;

- розпізнавання рукопису. Деякі програми розпізнавання тексту вміють перетворювати рукописний текст на комп'ютері у цифровий варіант;

- мовне розпізнавання. Деякі OCR-інструменти підтримують розпізнавання тексту у різних мовах, що робить їх універсальними для використання в різноманітних культурних та лінгвістичних середовищах;

- застосування в бізнесі. В бізнесі OCR може бути використаний для автоматизації операцій, обробки фінансової документації, розпізнавання карток візитів, ідентифікації товарів тощо;

- електронна книга та бібліотечні системи. OCR використовується для створення електронних книг, а також в діяльності бібліотек для перетворення тексту на фізичних документах у цифровий вигляд;

- розвиток штучного інтелекту. Технології OCR використовуються в розробці систем штучного інтелекту для поліпшення точності та швидкості розпізнавання тексту.

Програми розпізнавання тексту стають все більш важливими в сучасному цифровому світі, де швидкість та ефективність обробки інформації мають велике значення.

Технологія оптичного розпізнавання тексту (OCR) включає такі етапи:

1. Отримання зображення. Сканер зчитує документи та перетворює їх у двійкові дані. OCR аналізує відскановане зображення і класифікує світлі області як фон, а темні - як текст.
2. Попередня обробка. Щоб підготувати текст до розпізнавання, програмне забезпечення OCR очищає зображення та видаляє помилкові області. Застосовуються такі методи очищення: вирівнювання та усунення нахилу відсканованого документа для полегшення розпізнавання; згладжування контрасту або видалення плям цифрового зображення та згладжування крайових ефектів текстових зображень.
3. Стирання рамок та ліній на сканованому зображенні.
4. Розпізнавання шрифтів для багатомовної технології OCR.
5. Розпізнавання тексту.

Існує два основних типи алгоритмів OCR або програмних процесів, які використовує OCR для розпізнавання тексту: зіставлення шаблонів і виділення ознак.

Зіставлення шаблонів працює шляхом виділення зображення символу, який називається гліф, та порівняння його з аналогічним гліфом, що зберігається у пам'яті. Розпізнавання образу відбудеться лише в тому випадку, якщо шрифт і масштаб гліфа, що зберігається, збігаються зі шрифтом і масштабом відсканованого гліфа. Цей метод ефективний під час роботи зі сканами документів, набраних відомим шрифтом.

Виділення ознак розбиває або розкладає гліфи на такі ознаки, як лінії, замкнуті контури, напрямок ліній та перетин ліній. Потім ознаки використовуються для пошуку найкращої або найближчої відповідності відповідності серед різних гліфів, що зберігаються.

Після аналізу система перетворює вилучені текстові дані на комп'ютерний файл. Деякі системи OCR можуть створювати анотовані PDF-файли, які включають як попередню, так і наступну версію відсканованого документа.

В якості аналогу візьмемо програмну систему CuneiForm [7], яка дозволяє здійснювати оптичне розпізнавання текстів. Головне вікно цієї програми зображено на рисунку1.1.

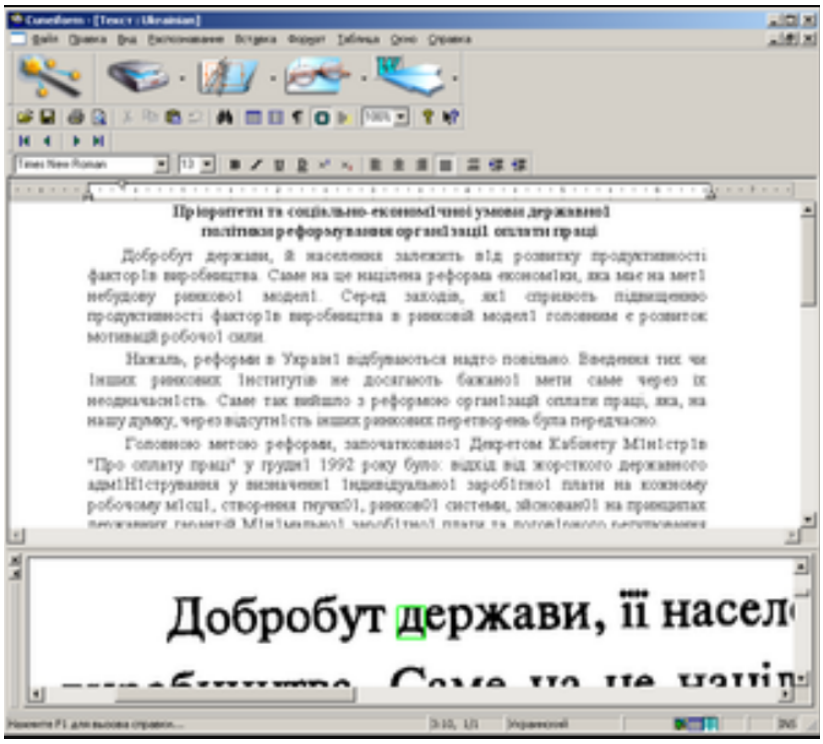

Рисунок 1.1 – Програмна система CuneiForm

CuneiForm, розроблений компанією Cognitive Technologies, представляє собою високоефективний інструмент оптичного розпізнавання символів, спроектований для перетворення зображень, отриманих із сканера або іншими методами, у текстовий формат [6].

Унікальною особливістю CuneiForm є його незалежність від шрифту, включаючи OmniFont. Алгоритми, вбудовані в цю систему, виходять за межі звичайних правил написання букв і базуються на їхній топології, що дозволяє впізнавати текст без необхідності визначення еталонів чи процедур навчання. CuneiForm демонструє вражаючу здатність розпізнавання будь-яких друкарських шрифтів, таких як у книгах, газетах, журналах, або виведених на друк лазерними та матричними принтерами, а також текстів, виведених на друкарських машинках тощо. Проте програма не розпізнає рукописний текст та декоративні шрифти, такі як готичний чи стилізований під рукописний.

CuneiForm володіє широким спектром можливостей для зручного використання розпізнаного тексту. Після процедури розпізнавання, отриманий текст може бути збережений у різних форматах, таких як RTF, HTML або простий текстовий формат. Крім того, є можливість передати розпізнаний текст безпосередньо до текстового процесора Word чи електронної таблиці Excel, спрощуючи подальшу роботу з ним.

У сучасному світі CuneiForm визнано як високоефективний інструмент для конвертації електронних документів і зображень у редагований формат. Ця система дозволяє зберігати структуру та шрифти оригіналу, надаючи можливість користувачеві робити це як у повністю автоматичному, так і у напівавтоматичному режимі. Таким чином, CuneiForm стає незамінним інструментом для тих, хто шукає ефективні рішення в галузі обробки і редагування текстової інформації.

Головним недоліком системи аналогу є невелика точність розпізнавання (90%). Ще одним недоліком даного програмного комплексу є його порівняно велика вартість, що робить його недоступним для більшості користувачів. Як і програма, що розробляється, даний аналог також написаний під операційну систему Windows.

1.4 Висновок до розділу 1

У даному розділі магістерської роботи була поставлена задача недостатньої достовірності сучасного процесу розпізнавання символів. Були розглянуті основні методи та програмні засоби, які виконують розпізнавання символів та визначені їх недоліки. В ході аналізу об’єкту проектування визначено: основні вимоги до інформаційної технології, вхідні та вихідні дані для програмних засобів. В результаті аналізу систем аналогів визначено їхні характеристики, переваги та недоліки. Для усунення цих недоліків було запропоновано використовувати метод розпізнавання символів, заснований на використанні штучних нейронних мереж та так званих лінійних сенсорів. На основі даного методу потрібно розробити інформаційну технологію, реалізувати її програмно та перевірити чи будуть задоволені вимоги до програмного засобу та усунуті вказані недоліки.

2 РОЗРОБКА ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ОПТИЧНОГО РОЗПІЗНАВАННЯ ТЕКСТУ

2.1 Обґрунтування вибору типу нейронної мережі

Найбільш широко досліджуваною і використовуваною нейронною мережею є багатошаровий персептрон (Multi-Level Perceptron – MLP) [7]. Така структура, що навчається за допомогою методу зворотного поширення помилки, є однією з найбільш популярних і універсальних форм нейронних мереж-класифікаторів і однією з найбільш часто використовуваних для розпізнавання символів та тексту.

ШНМ (штучні нейромережі) – це математична модель функціонування традиційних для живих організмів нейромереж, які є мережею нервових клітин. Як і в біологічному аналогу, у штучних мережах основним елементом виступають нейрони, з'єднані між собою і утворюють шари, число яких може бути різним залежно від складності нейромережі та її призначення (розв'язуваних завдань).

Мабуть, найпопулярнішим завданням нейромереж є розпізнавання візуальних образів. Сьогодні створюються мережі, в яких машини здатні успішно розпізнавати символи на папері та банківських картках, підписи на офіційних документах, детектувати об'єкти тощо. Ці функції дозволяють суттєво полегшити працю людини, а також підвищити надійність та точність різних робочих процесів за рахунок відсутності можливості припущення помилки через людський фактор.

Нейронна мережа (НМ) – це математична модель у вигляді програмного та апаратного втілення, що будується на принципах функціонування біологічних нейромереж. Сьогодні такі мережі активно використовують у практичних цілях за рахунок можливості не лише розробки, а й навчання. Їх застосовують для прогнозування, розпізнавання образів, машинного перекладу, розпізнавання аудіо тощо.

Звичайною часто називають повнозв'язкову нейронну мережу. У ній кожен вузол (крім вхідного та вихідного) виступає як входом, так і виходом, утворюючи прихований шар нейронів, і кожен нейрон наступного шару з'єднаний з усіма

нейронами попереднього. Входи подаються з терезами, які в процесі навчання налаштовуються і не змінюються надалі. При цьому кожен нейрон має поріг активації, після проходження якого він приймає одне з двох можливих значень: -1 або 1, чи 0 або 1.

Згорткова нейронна мережа (ЗНМ) має спеціальну архітектуру, яка дозволяє їй максимально ефективно розпізнавати образи. Сама ідея ЗНМ ґрунтується на чергуванні згорткових та субдискретизуючих шарів (pooling), а структура є односпрямованою. СНР отримала свою назву від операції згортки, яка передбачає, що кожен фрагмент зображення буде помножений на ядро згортки поелементно, при цьому отриманий результат повинен підсумовуватися і записатися в схожу позицію вихідного зображення. Така архітектура забезпечує інваріантність розпізнавання щодо зсуву об'єкта, поступово укрупнюючи «вікно», на яке «дивиться» згортка, виявляючи дедалі більші структури та патерни в зображенні.

Робота із зображеннями – важлива сфера застосування технологій Deep Learning. Глобально всі зображення з усіх камер світу становлять бібліотеку неструктурованих даних. Задіявши нейромережі, машинне навчання та штучний інтелект, ці дані структурують та використовують для виконання різних завдань: побутових, соціальних, професійних та державних, зокрема, забезпечення безпеки.

Основою всіх архітектур для відеоспостереження є аналіз, першою фазою якого буде розпізнавання зображення (об'єкта). Потім штучний інтелект за допомогою машинного навчання розпізнає дії та класифікує їх.

Для того щоб розпізнати зображення, нейронна мережа має бути раніше навчена на даних. Це дуже схоже на нейронні зв'язки в людському мозку — ми маємо певні знання, бачимо об'єкт, аналізуємо його та ідентифікуємо.

Нейромережі вимогливі до розміру та якості датасету, на якому вона навчатиметься. Датасет можна завантажити з відкритих джерел або зібрати самостійно

На практиці означає, що до певної межі чим більше прихованих шарів у нейронній мережі, тим точніше буде розпізнане зображення. Як це реалізується?

Картинка розбивається на маленькі ділянки, аж до кількох пікселів, кожен із яких буде вхідним нейроном. За допомогою синапсів сигнали передаються від одного шару до іншого. Під час цього процесу сотні тисяч нейронів із мільйонами параметрів порівнюють отримані сигнали з уже обробленими даними.

Простіше кажучи, якщо ми просимо машину розпізнати фотографію кішки, ми розіб'ємо фото на маленькі шматочки і порівнюватимемо ці шари з мільйонами вже наявних зображень кішок, значення ознак яких мережа вивчила.

У якийсь момент збільшення числа шарів призводить до просто запам'ятовування вибірки, а не до навчання. Далі – за рахунок хитрих архітектур.

Нейронна мережа для розпізнавання зображень – це, мабуть, найпопулярніший спосіб застосування НМ. При цьому незалежно від особливостей розв'язуваних завдань вона працює по етапах, найважливіші серед яких розглянемо нижче.

Як образи, що розпізнаються, можуть виступати різні об'єкти, включаючи зображення, рукописний або друкований текст, звуки і багато іншого. При навчанні мережі їй пропонуються різні зразки з позначкою того, якого саме типу їх можна віднести. Як зразок застосовується вектор значень ознак, а сукупність ознак у умовах повинна дозволити однозначно визначити, з яким класом образів має справу НМ.

Важливо під час навчання навчити мережу визначати як достатню кількість і значення ознак, щоб видавати хорошу точність на нових зображеннях, а й не перенавчитися, тобто, надмірно не «підлаштуватися» під навчальну вибірку з зображень. Після завершення правильного навчання НМ повинна вміти визначати образи (тих класів), із якими вона мала справи у процесі навчання.

Важливо враховувати, що вихідні дані для нейромережі повинні бути однозначними і несуперечливими, щоб не виникали ситуації, коли НМ видаватиме високі ймовірності приналежності одного об'єкта до кількох класів.

Виділяють кілька різних архітектур штучних НМ, у тому числі нейромереж для розпізнавання зображень:

- Багатошаровий перцептрон. Будується з 3+ шарів та застосовує нелінійну функцію активації для класифікації даних;

- Згорткова НМ. Містить згорткові шари;

- Рекурсивна НМ. Глибока НМ, яка формується застосуванням одних наборів терезів рекурсивно над структурою для скалярних або структурованих передбачень;

- Рекурентна НМ. Варіант НМ, де зв'язки між нейронами є спрямованими циклами;

- Мережа довгої короткострокової пам'яті (МДКП) – вид рекурентної НМ, що дозволяє максимально точно моделювати тимчасові послідовності, а також характерні для них залежності у довгостроковій перспективі;

- Sequence-to-sequence модель. Складається з 2х рекурентних НМ, які виконують функції кодувальника та декодера;

- Неглибокі НМ. Також користуються великою популярністю, наприклад, групи неглибоких двошарових моделей можна використовувати для представлення шарів векторами.

Розпізнавання зображень за допомогою нейронних мереж можливе лише за допомогою спеціального навчання, що є процесом, спрямованим на налаштування параметрів НМ.

Є кілька способів навчити нейромережу.

При навчанні НМ для розпізнавання образів з учителем є вибірка з істинними відповідями питання, що зображено малюнку – мітками класів. Нейромережі подають на вхід ці зображення, після чого обчислюється помилка, що порівнює вихідні значення з мітками класів. Залежно від ступеня та характеру невідповідності передбачення НМ, її ваги коригуються, відповіді НМ підлаштовуються під справжні відповіді, поки помилка не стане мінімальною.

У цьому випадку у навчальної вибірки немає міток класів, і перед НМ стоїть завдання знайти заздалегідь не відомі відповіді. Нейронна мережа намагається самостійно знайти закономірності у даних, витягуючи корисні ознаки та аналізуючи їх. Наприклад, кластеризація — найпоширеніша завдання навчання без вчителя. Алгоритм підбирає схожі дані, знаходячи загальні ознаки і групують їх разом.

У навчанні без вчителя складно обчислити точність алгоритму, оскільки в даних відсутні «правильні відповіді» чи мітки. Але розмічені дані буває складно чи

надто дорого отримати. У таких випадках, надаючи моделі свободу дій для пошуку залежностей, можна отримати певний результат.

Навчальна вибірка містить як розмічені, і нерозмічені дані. Цей метод особливо корисний, коли розмітити всі об'єкти – трудомістке завдання. Тим не менш, нейронна мережа може отримати інформацію з невеликої частки розмічених даних і покращити точність передбачень у порівнянні з моделлю, що навчається виключно на нерозмічених даних.

Навчання із підкріпленням (reinforcement learning) діє за принципом отримання зворотного зв'язку - нагороди за певні дії.

Важливою характеристикою будь-якої нейронної мережі є її властивість до навчання. Процес навчання представляє собою складну процедуру, яка спрямована на оптимізацію ваг і порогів з метою мінімізації розбіжності між бажаними (цільовими) та фактично отриманими векторами на виході мережі. В своїй книзі Розенблат висловлював спроби класифікації різних алгоритмів навчання персептрону, охрещуючи їх системами підкріплення.

Для багатошарових мереж був введений градієнтний алгоритм навчання з учителем, який передає сигнал помилки, обчислений на виходах персептрона, до його входів, шар за шаром. Цей метод, який був запропонований Д. Румельхартом та іншими вченими, на сьогодні є найбільш популярним для навчання багатошарових персептронів. Відзначено, що його основною перевагою є можливість навчати всі шари нейронної мережі та легкість локального обчислення. Тим не менш, цей метод вимагає значних обчислювальних ресурсів та диференційованості передавальної функції нейронів, що передбачає перехід до роботи з неперервними значеннями на вході персептрона, відмовляючись від бінарного сигналу [5].

Через вищезазначені причини для реалізації інформаційної технології розпізнавання символів було обрано нейронну мережу багатошаровий персептрон, яка навчається за методом зворотного поширення помилки. Такі мережі часто так і називають – мережі зворотного поширення помилки.

2.2 Математична модель багатошарового персептрона

Багатошарові персептрони представляють собою конкретний тип нейронних мереж прямого поширення. У цих мережах вхідний сигнал поширюється в прямому напрямку, шар за шаром. Загальні характеристики багатошарового персептрону включають наступне:

- множину вхідних вузлів, які утворюють вхідний шар. Ці вхідні вузли служать для прийому вхідної інформації, яка подається до мережі;

- один або декілька прихованих шарів обчислювальних нейронів. Кожен нейрон у прихованому шарі здійснює обчислення на основі вхідних сигналів і ваг, пов'язаних з кожним вхідним вузлом, і передає результат вихідному шару;

- один вихідний шар нейронів, який відповідає за вивід результату мережі. Кожен нейрон в вихідному шарі генерує вихідний сигнал на основі обчислень, проведених в прихованих шарах.

Багатошарові персептрони використовуються для вирішення широкого спектру завдань, таких як класифікація, регресія, апроксимація функцій та розпізнавання образів. Однією з ключових переваг таких мереж є їхній потенціал для вивчення складних нелінійних залежностей в даних.

Багатошаровий персептрон являє собою узагальнення одношарового персептрона Розенблатта. Прикладом багатошарового персептрона є наступна модель нейронної мережі, зображена на рисунку 2.1.

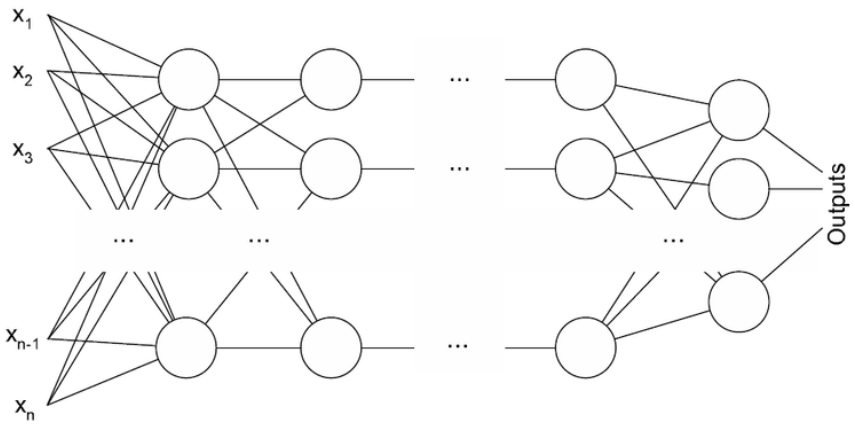

Рисунок 2.1 – Архітектура багатошарового персепторна

Кількість вхідних та вихідних елементів в середині багатошарового персептрона визначається умовами завдання. Сумніви можуть виникнути щодо того, які вхідні значення використовувати, а які ні. Питання про те, скільки використовувати проміжних шарів і елементів в них, поки зовсім неясне. В якості початкового наближення, зазвичай беруть один проміжний шар, при цьому число елементів у ньому рівне півсумі числа вхідних і вихідних елементів.

Багатошарові персептрони, як потужний клас нейронних мереж, демонструють вражаючу ефективність в розв'язанні різноманітних складних завдань. Ці мережі виокремлюються та визначаються наступними трирівневими характеристиками:

- глибока структура - багатошарові персептрони характеризуються глибокою структурою, що включає в себе кілька прихованих шарів між вхідним і вихідним шарами. Це дозволяє їм автоматично визначати та використовувати складні функціональні залежності в даних. Глибокі архітектури забезпечують багатошаровим персептронам високий рівень абстракції і здатність розпізнавати властивості, які можуть бути невидимими для менш глибоких моделей;

- процес навчання - однією з ключових рис багатошарових персептронів є їх здатність до автоматичного навчання. Процес навчання включає налаштування ваг та порогів мережі з метою мінімізації різниці між очікуваними та фактичними виходами. Ця здатність до адаптації робить їх ефективними інструментами для розв'язання різноманітних завдань, включаючи класифікацію, регресію та інші;

- різноманітні завдання - багатошарові персептрони успішно використовуються в різноманітних областях, таких як комп'ютерне зорове сприйняття, обробка природних мов, медична діагностика, фінансова аналітика та багато інших. Їхні здатності розв'язувати завдання з великою кількістю даних та складних залежностей роблять їх популярними серед інструментів штучного інтелекту.

Ці характеристики роблять багатошарові персептрони важливим елементом у світі глибокого навчання та штучного інтелекту загалом.

Кожен нейрон у мережі обладнаний нелінійною функцією активації. Важливо відзначити, що ця нелінійна функція повинна мати гладкий характер, що означає, що

вона є диференційованою у всіх точках. Це відрізняє її від жорсткої порогової функції, що використовується в оригінальному персептроні Розенблатта.

Найпопулярнішою формою функції, що задовольняє цій вимозі, є сигмоїдальна. Прикладом сигмоїдальної функції може слугувати логістична функція, що задається наступним виразом:

$$OUT = \frac{1}{1 + \exp\left(- \alpha Y\right)} \tag{2.1}$$

де α - параметр нахилу сигмоїдальної функції. Змінюючи цей параметр, можна побудувати функції з різним нахилом.

Наявність нелінійності відіграє дуже важливу роль, тому що в іншому випадку відображення «вхід-вихід» мережі можна звести до звичайного одношарового персептрону.

Багатошаровий персептрон, будучи розгалуженою формою штучної нейронної мережі, включає в себе не тільки вхідний та вихідний шари, але і один або кілька шарів прихованих нейронів, які знаходяться між вхідним та вихідним шарами. Ці приховані шари виконують ключову роль у здатності мережі до ефективного вирішення складних завдань.

Приховані шари не входять безпосередньо в систему передачі вхідних або вихідних сигналів, але, натомість, представляють собою внутрішній шар обчислювальних нейронів. Ці нейрони можуть взаємодіяти із зовнішніми сигналами, навчаючись виокремлювати та враховувати найбільш важливі ознаки чи патерни з вхідного образу. Таким чином, приховані шари дозволяють мережі адаптуватися до різноманітних структур та взаємозв'язків у вхідних даних, що робить їхнє застосування ефективним для розпізнавання та класифікації складних патернів.

Ступінь зв'язності у багатошаровому персептроні є визначальною властивістю, яка забезпечує його здатність ефективно опрацьовувати інформацію. Це досягається завдяки синаптичним з'єднанням, що забезпечують взаємодію між нейронами різних

шарів мережі. Зміна рівня зв'язності вимагає налаштування безлічі синаптичних з'єднань або їхніх вагових коефіцієнтів.

Синаптичні з'єднання визначають ступінь взаємодії між окремими нейронами та регулюють передачу сигналів через мережу. Зміна рівня зв'язності може бути реалізована шляхом налаштування вагових коефіцієнтів або створення нових синаптичних з'єднань. Ця гнучкість в структурі дозволяє багатошаровому персептрону ефективно адаптуватися до різноманітних завдань та забезпечує високий ступінь обчислювальної потужності [9].

Для навчання багатошарового персептрону виберемо алгоритм зворотного поширення помилки.

Алгоритм зворотного поширення помилки є одним з найбільш поширених методів для навчання багатошарових персептронів. Цей метод ґрунтується на принципі градієнтного спуску та використовується для оптимізації ваг і порогів нейронів з метою мінімізації помилки між фактичними та очікуваними виходами мережі.

Процес навчання за алгоритмом зворотного поширення полягає в тому, що спочатку проводиться передача сигналу вперед через мережу для отримання прогнозованого виходу. Потім обчислюється помилка, яка є різницею між цільовим вихідним сигналом і фактичним виходом мережі. Далі ця помилка поширюється назад через мережу, і для кожного нейрона визначається внесок у загальну помилку.

Градієнти, отримані в результаті цього процесу, використовуються для корекції ваг і порогів нейронів з метою покращення точності прогнозу та збільшення ефективності мережі. Важливо відзначити, що алгоритм зворотного поширення дозволяє мережі адаптуватися до різних умов і вирішувати складні завдання завдяки ітеративному процесу корекції параметрів.

У мережі є множина входів x1, ..., xn, множина виходів Outputs і множина внутрішніх вузлів. Застосуємо нумерацію всіх вузлів (з входами та виходами включно) числами від 1 до N (нумерація має бути наскрізною і не залежати від топології шарів). Позначимо вагу зв'язку, що з'єднує i-й та j-й вузли як w{i,j}, а вихід i-го вузла як o_i. Якщо ж навчальний приклад відомий заздалегідь (ми знаємо

правильні відповіді мережі t_k, при цьому k є Outputs), то функція помилки, що отримана методом найменших квадратів, має вигляд:

$$E(\{w_{i,j}\}) = \frac{1}{2} \sum_{k \in Outputs} (t_k - o_k)^2 \tag{2.2}$$

Як змінити ваги? Ми впроваджуватимемо стохастичний градієнтний спуск, що означає, що ваги будуть коригуватися після кожного навчального прикладу, і, таким чином, ми будемо «рухатися» у багатовимірному просторі ваг. Щоб досягти мінімізації помилки, необхідно «рухатися» в напрямку, протилежному градієнту. Тобто, на основі кожного набору правильних відповідей, ми будемо додавати до кожної ваги $w_{\{i,j\}}$.

$$\Delta w_{i,j} = -\eta \frac{\partial E}{\partial w_{i,j}}, \tag{2.3}$$

де $0 < \eta < 1$ — множник, який регулює швидкість «руху».

Похідна обчислюється наступним чином. Припустимо, спочатку j належить до виходів (Outputs), що означає, що вага, яка нас цікавить, входить в нейрон останнього рівня. На початку відзначимо, що $w_{\{i,j\}}$ впливає на вихід мережі лише як частина суми. $S_j = \sum w_{\{i,j\}} x_i$, де сума обчислюється по входах j-го вузла. Тому

$$\frac{\partial E}{\partial w_{i,j}} = \frac{\partial E}{\partial S_j} \frac{\partial S_j}{\partial w_{i,j}} = x_i \frac{\partial E}{\partial S_j}. \tag{2.4}$$

Аналогічно, параметр S_j має вплив на загальну помилку тільки в межах виходу j-го вузла o_j (а це є вихід всієї мережі). Тому

$$\frac{\partial E}{\partial S_j} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial S_j} = \left(\frac{\partial}{\partial o_j}\frac{1}{2}\sum_{k \in Outputs}(t_k - o_k)^2\right)\left(\frac{\partial \sigma(S_j)}{\partial S_j}\right) = \left(\frac{1}{2}\frac{\partial}{\partial o_j}(t_j - o_j)^2\right)(o_j(1-o_j)) = -o_j(1-o_j)(t_j-o_j) \quad (2.5)$$

Якщо ж j-й вузол – знаходиться не на останньому рівні, то у нього теж є виходи. Отже позначимо їх як Children (j). Тоді:

$$\begin{gathered}\frac{\partial E}{\partial S_j} = \sum_{k \in Children(j)} \frac{\partial E}{\partial S_k}\frac{\partial S_k}{\partial S_j} \\ \frac{\partial S_k}{\partial S_j} = \frac{\partial S_k}{\partial o_j}\frac{\partial o_j}{\partial S_j} = w_{i,j}\frac{\partial o_j}{\partial S_j} = w_{i,j}o_j(1-o_j)\end{gathered} \quad (2.6)$$

Оскільки ми опанували метод обчислення корекцій для вузлів останнього рівня і можемо виразити корекцію для вузла на нижчому рівні через корекції вищого рівня, зараз можемо розглядати створення алгоритму навчання. Цей алгоритм відомий як алгоритм зворотного поширення помилки (англ. backpropagation) із-за особливостей обчислення корекцій, які він використовує.

Підведемо підсумок:

- Для вузла, що знаходиться на останньому рівні:

$$\delta_j = -o_j(1-o_j)(t_j - o_j) \quad (2.7)$$

- Для внутрішнього вузла нейронної мережі:

$$\delta_j = -o_j(1-o_j)\sum_{k \in Outputs(j)} \delta_k w_{j,k} \quad (2.8)$$

- Для усіх вузлів нейронної мережі:

$$\Delta w_{i,j} = -\eta \delta_j x_i \quad (2.9)$$

На вхід алгоритму, окрім вже зазначених параметрів, також необхідно подавати структуру мережі в якомусь форматі. На практиці, дуже ефективні результати

показують мережі з відносно простою структурою, які складаються з двох рівнів нейронів: прихованого рівня (hidden units) і рівня виходу (output units). Кожен вхід мережі з'єднаний з усіма нейронами прихованого рівня, а результат роботи кожного прихованого нейрона подається на вхід кожному нейрону рівня виходу. У такому випадку, досить подавати на вхід кількість нейронів прихованого рівня. [9].

2.3 Структура інформаційної технології оптичного розпізнавання тексту

Ідея, на основі якої буде функціонувати програма, базується на використанні так званих сенсорів. Припустимо, у нас є зображення з літерою довільного розміру. При такому підході ми утворюємо вхідний вектор, використовуючи не значення пікселів зображення, а значення сенсорів. Що за сенсори? Сенсори представлені набором ліній з довільною величиною і напрямком (рис. 2.2). Будь-який сенсор матиме активоване значення ("0,5" у вхідному векторі), якщо він перетинає літеру і дезактивоване значення ("-0.5" у вхідному векторі), якщо він не перетинає літеру. Розмір вхідного вектору дорівнюватиме кількості сенсорів. До речі, ми можемо використовувати набір коротких горизонтальних і вертикальних сенсорів для досягнення такого ж ефекту, як у випадку використання значень пікселів для невеликих зображень.

Рисунок 2.2 – Набір сенсорів для розпізнавання літери А

Перевага цього методу в тому, що ми можемо навчати нейронну мережу на великих зображеннях навіть при невеликій кількості сенсорів. Зміна розміру зображення літери на 75x75 (або навіть 150x150) пікселів не призведе до поганої якості зображення, тому розпізнавання буде набагато простішим. Але, з іншого боку, завжди можна легко змінити набір наших сенсорів, так як вони визначені як лінії з

двома координатами. І ще великий плюс у тому, що ми можемо спробувати створити досить невеликий набір сенсорів, який буде в змозі розпізнати весь навчальний набір, використовуючи лише найбільш істотні особливості літер.

Але, є деякі недоліки. Описаний підхід може бути застосований лише до задачі розпізнавання символів. Неможливо, розпізнати складні образи, тому що в такому випадку ми повинні будемо використовувати занадто багато сенсорів. Але, оскільки ми проводимо дослідження в області розпізнавання символів, цей метод не буде вимагати багато ресурсів. Існує ще одне питання, яке набагато складніше і вимагає більше досліджень. Як створити набір сенсорів? Вручну чи випадково? Як впевнитись, що набір є оптимальним?

Ми можемо використовувати наступний підхід для генерації сенсорів: по-перше, ми випадковим чином генеруємо великий набір сенсорів, і тоді ми виберемо певну кількість кращих сенсорів. Як визначити, чи сенсор є підходящим чи ні? Ми постараємося використовувати ентропію, яка добре відома нам з теорії інформації. Приклад представлення образів і сенсорів зображено на рисунку 2.3.

	0	1	2
0	11101	00000	00000
1	11111	00000	01011
2	11101	00000	00000
3	11111	00010	11111
4	10101	00000	00000

Рисунок 2.3 – Представлення образів та набору сенсорів, за допомогою яких відбудеться розпізнавання образів

Ось таблиця, яка містить деякі навчальні дані. Ми можемо побачити п'ять типів об'єктів, які представлені рядками і три сенсори, які представлені стовпцями. Кожен об'єкт має п'ять різних варіантів. Опишемо, наприклад, перше значення в таблиці: "11101". Це означає, що перший сенсор перетинає перший варіант першого об'єкта, він також перетинає другий, третій і п'ятий варіанти, але не перетинає четвертий варіант. Зрозуміло? Добре, давайте поглянемо на п'ятий рядок, перший стовпець:

сенсор перетинає перший, третій і п'ятий варіанти, і це не перетинає другий і четвертий варіанти.

Ми будемо використовувати два поняття: внутрішню і зовнішню ентропію. Внутрішня - це ентропія певного сенсора для певного об'єкта. Внутрішня ентропія показує нам, наскільки добре певний сенсор підходить для розпізнавання певного об'єкту, а значення має бути мінімальним. Давайте подивимося на другий рядок і перший стовпець, де міститься значення "11111". Ентропія набору буде 0. Це добре, тому що цей сенсор на 100% вказує, що ми працюємо з другим об'єктом, оскільки сенсор має однакове значення для всіх варіантів цього об'єкта. Те ж саме вірно і для другого рядка і другого стовпця: "00000". Його ентропія теж дорівнює 0. Але, давайте подивимося на п'ятий рядок і перший стовпець: "10101". Ентропія набору дорівнює 0,971, і це погано, тому що сенсор не впевнений в цьому об'єкті. Зовнішня ентропія обчислюється для всього стовпця, і чим ближче її значення до "1", тим краще. Чому? Якщо зовнішня ентропія мала, то сенсор негодящий, тому що він не може розділити образи. Найкращий сенсор повинен бути активований для однієї половини всіх об'єктів і дезактивований для іншої. Отже, ось остаточна формула для розрахунку корисності сенсорів: корисність = Зовн.Ентропія * (1 – СередняВнутр.Ентропія). Середня внутрішня ентропія просто – це сума внутрішніх ентропій сенсора для всіх об'єктів, поділена на кількість об'єктів.

Таким чином, використовуючи цю ідею, спочатку ми повинні згенерувати випадковим чином великий набір сенсорів, наприклад 500. Тоді ми повинні згенерувати тимчасові навчальні входи, використовуючи ці сенсори. На підставі цих даних, ми можемо відфільтрувати початкову кількість сенсорів, наприклад, ми можемо залишити 100 найкорисніших сенсорів. Процедура фільтрації зменшить кількість навчальних даних і входів нейронної мережі. Потім, з відфільтрованими навчальними даними, ми зможемо продовжити підготовку нашої мережі.

Структура інформаційної технології розпізнавання тексту на основі лінійних сенсорів та нейронної мережі багатошаровий персептрон, детально описана у попередніх підпунктах, зображена на рисунку 2.4.

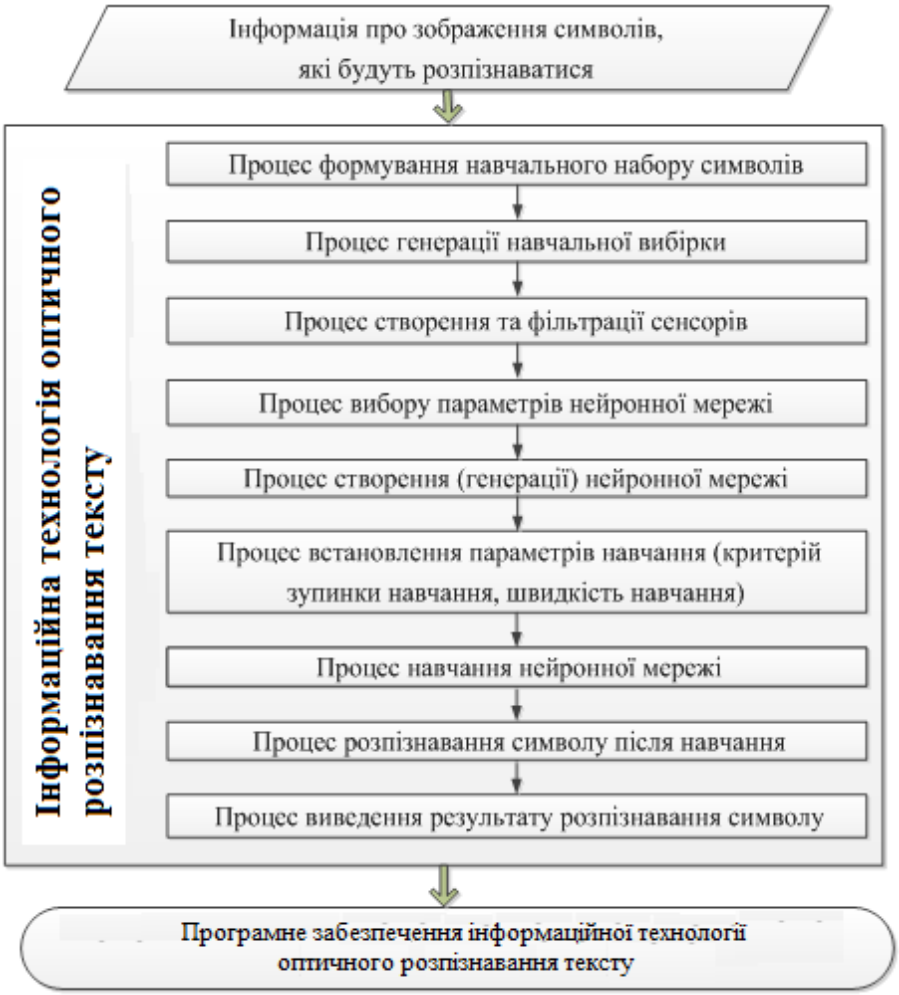

Рисунок 2.4 - Структура інформаційної технології розпізнавання тексту

Вхідною інформацією для даної інформаційної технології є зображення текстових символів, які будуть розпізнаватися. Для роботи інформаційної технології розпізнавання тексту спочатку потрібно сформувати навчальний набір символів, які будуть розпізнаватись. Після цього відбувається генерація навчальної вибірки шляхом спотворення кожного еталонного символа операціями невеликого зсуву, незначного масштабування та повороту на невеликі кути. Далі відбувається створення та фільтрація сенсорів. Після цього проводиться вибір параметрів нейронної мережі (кількість шарів, кількість нейронів у шарі, вид функції активації і т.п.). Далі відбувається створення (генерація) самої нейронної мережі з рандомними значеннями ваг. Потім проводиться встановлення параметрів навчання (критерій зупинки навчання, швидкість навчання і т.п.). І вже після цього потрібно провести навчання нейронної мережі за алгоритмом зворотного поширення помилки. На

навченій мережі вже можна проводити розпізнавання символу. Після цього потрібно вивести результат розпізнавання.

Таким чином, розроблена структура інформаційної технології оптичного розпізнавання тексту на основі лінійних сенсорів та нейронної мережі багатошаровий персептрон може бути використана для подальшої розробки програмних засобів.

2.4 Алгоритм функціонування інформаційної технології оптичного розпізнавання тексту

Вхідною інформацією для даної програми є зображення символів, які будуть розпізнаватися. Це багато відомих типів шрифтів, які наша нейронна мережа буде навчатись розпізнавати. Серед доступних в програмі шрифтів є такі: Arial, Courier, Tahoma, Times New Roman, Verdana. Для кожного з шрифтів можна вибрати його тип: стандартний або курсивний. Для роботи інформаційної технології розпізнавання символів спочатку потрібно сформувати навчальний набір символів (вершина 1), які будуть розпізнаватись. Після цього відбувається генерація навчальної вибірки (вершина 2) шляхом спотворення кожного еталонного символа операціями невеликого зсуву, незначного масштабування та повороту на невеликі кути. Далі відбувається створення та фільтрація сенсорів (вершини 3,4). Після цього проводиться вибір кількості шарів нейронної мережі (вершина 5) та створення (генерація) самої нейронної мережі (вершина 6) з рандомними значеннями ваг. Потім проводиться вибір критерію зупинки навчання (вершина 7), після чого встановлюється або значення порогу похибки (вершина 8) або значення кількості епох навчання (вершина 9). Швидкість навчання встановлюється у вершині 10. І вже після цього потрібно провести навчання нейронної мережі за алгоритмом зворотного поширення помилки (вершина 11). На навченій мережі вже можна проводити розпізнавання символу (вершина 12). Після цього потрібно вивести результат розпізнавання (вершина 13). Так можна розпізнавати багато символів , доки не буде обрано завершення роботи програми (рис. 2.5).

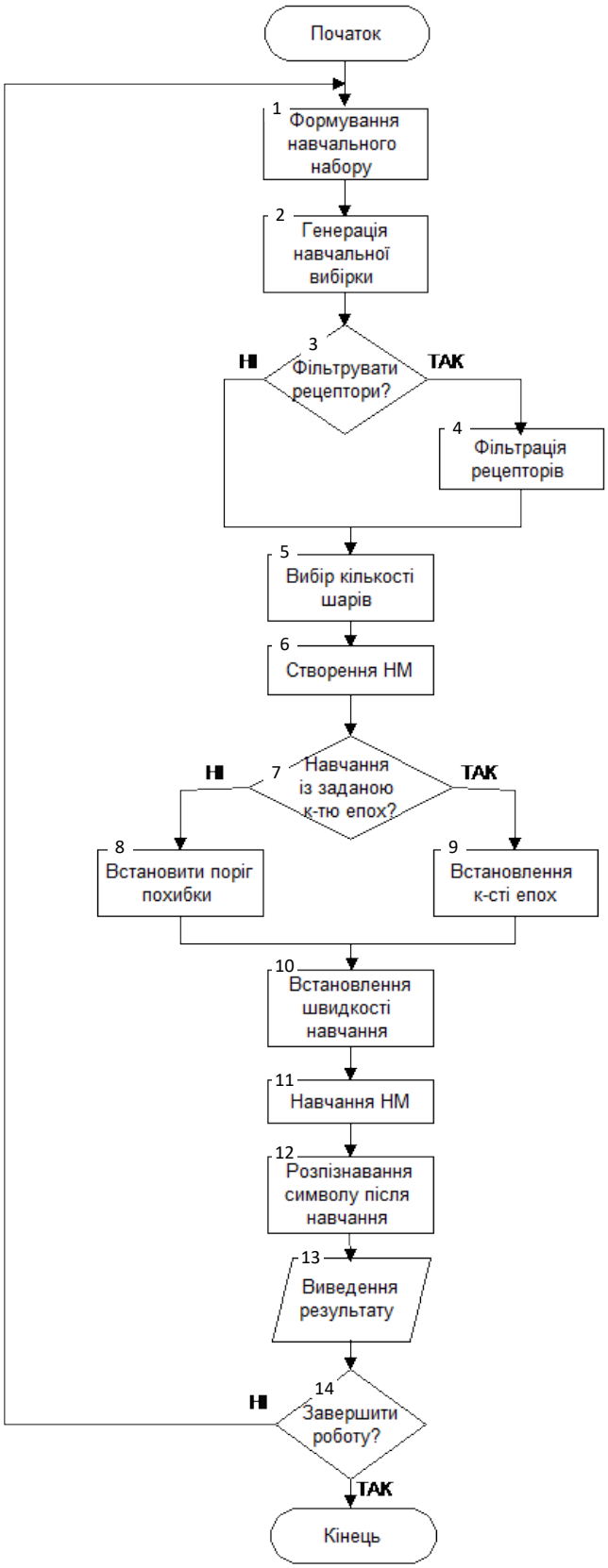

Рисунок 2.5 – Алгоритм роботи інформаційної технології оптичного розпізнавання тексту

2.5 Розробка UML-діаграми класів

UML (англ. Unified Modeling Language) — це уніфікована мова моделювання, що використовується в парадигмі об'єктно-орієнтованого програмування і є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення.

UML представляє собою мову широкого профілю та відкритий стандарт, який використовує графічні позначення для створення абстрактної моделі системи, відомої як UML-модель. Задачі, які стоять перед UML, включають визначення, візуалізацію, проектування та документування програмних систем.

Важливо підкреслити, що UML не є мовою програмування, але в засобах виконання UML-моделей можлива генерація вихідного коду.

UML може бути застосовано на всіх етапах життєвого циклу аналізу бізнес-систем і розробки прикладних програм. Різні види діаграм, які підтримуються UML, та широкі можливості представлення різних аспектів системи роблять UML універсальним інструментом для опису як програмних, так і ділових систем.

Діаграми в рамках мови UML надають можливість зображати систему, як бізнес-орієнтовану, так і програмну, у формі, яку можна легко трансформувати в програмний код. Однією з ключових причин використання мови UML є сприяння ефективній комунікації між розробниками.

UML була спеціально створена для оптимізації процесу розробки програмних систем, сприяючи значному підвищенню продуктивності та помітному покращенню якості кінцевого продукту. Використання UML у багатьох успішних програмних проектах підкреслює її ефективність.

Особливо слід відзначити, що інструменти автоматичної генерації коду, що входять в арсенал UML, дозволяють конвертувати моделі, створені мовою UML, безпосередньо у вихідний код об’єктно-орієнтованих мов програмування. Це різко прискорює процес розробки, роблячи його більш ефективним та менш витратним.

CASE-засоби, що використовуються для автоматизації процесів аналізу та проектування, в основному, мають вбудовану підтримку мови моделювання UML

(Unified Modeling Language). Застосування UML виявляється дуже ефективним на етапі розробки програмного забезпечення.

Моделі, створені в мові UML, стають потужним інструментом для спрощення процесу написання коду. Це дозволяє розробникам зосередитися безпосередньо на реалізації системи, оскільки UML надає абстракції та графічні позначення, які полегшують розуміння та взаємодію зі структурою та логікою проекту.

Використання діаграм у мові UML вносить суттєвий внесок у супроводжуваність проекту. Діаграми дозволяють чітко візуалізувати різні аспекти системи, що сприяє як зрозумілості для команди розробників, так і полегшенню процесу створення документації.

У випадку модифікації системи об'єктно-орієнтований підхід, який використовується в UML, робить процес додавання нових об'єктів чи вилучення застарілих відносно легким. Цей підхід дозволяє внесення змін без суттєвої перебудови структури системи, забезпечуючи тим самим її життєздатність. Використання побудованої моделі при модифікаціях системи дозволяє ефективно вирішувати можливі проблеми та уникати небажаних наслідків змін.

Переваги використання Unified Modeling Language (UML) включають:

- широкий опис системи. UML дозволяє описати систему з різних точок зору та враховує різні аспекти її поведінки. За допомогою різних видів діаграм можна моделювати відносини між об'єктами, їхню структуру, динаміку і багато іншого;

- простоту читання. Діаграми UML є порівняно простими для читання, особливо після короткого періоду знайомства з їхнім синтаксисом. Це дозволяє команді розробників легко розуміти та взаємодіяти з моделями;

- розширюваність. UML розширюється, і користувачі можуть вводити власні текстові та графічні стереотипи, адаптуючи його до конкретних потреб проекту. Це робить UML гнучким і застосовним не тільки в сфері програмної інженерії, але й у різних галузях;

- широке застосування. UML отримав широке поширення та постійно розвивається. Велика кількість інструментів і підтримка з боку розробників сприяють його активному використанню в індустрії програмного забезпечення.

Зазначені переваги дозволяють UML залишатися потужним та важливим інструментом для моделювання систем, надаючи командам розробників ефективний засіб спілкування та оптимізації процесу розробки програмного забезпечення.

Недоліки використання Unified Modeling Language (UML) включають:

- зайву складність. UML включає в себе значну кількість діаграм та конструкцій, які в певних випадках можуть бути надто складними або зайвими для конкретного проекту. Деякі з них можуть залишатися невикористаними у реальній розробці, що призводить до перевантаження моделі непотрібною інформацією;

- неточність і візуальну неоднорідність. UML може бути відзначено відсутністю конкретних визначень та точних специфікацій для деяких елементів, що може призводити до неоднозначності в трактуванні деяких концепцій. Візуальна неоднорідність може виникати через різні стилі та інтерпретації, що робить читання та редагування документів UML менш інтуїтивно зрозумілими;

- загальність для багатьох галузей. UML була розроблена для застосування в різних галузях розробки, що може призводити до загальності в описі конкретних особливостей деяких галузей. Внаслідок цього деякі розробники можуть вважати, що деякі аспекти моделювання не відображають конкретні вимоги їхніх проектів.

Хоча UML є потужним інструментом, його недоліки слід враховувати, спираючись на конкретні потреби та характеристики проекту, щоб використання мови моделювання було більш ефективним і зручним.

Діаграма класів (Class diagram) в рамках моделювання програмного забезпечення відображає онтологію домену, що істотно еквівалентна структурі інформаційної моделі, розробленої методом С. Шлєєра і С. Мелора. Вона визначає склад класів об'єктів та їхні зв’язки, сприймаючи змістовну роль у визначенні структури програмного продукту.

Діаграма класів зазвичай створюється за допомогою графічних елементів, де класи представлені поділеними на три частини прямокутниками, а їх зв’язки відображаються лініями, які з’єднують ці прямокутники. Такий підхід відповідає візуальному представленню понять і взаємозв'язків між ними.

Верхня частина прямокутника, що представляє клас, є обов'язковою і містить ім'я класу. Друга і третя частини прямокутника визначають відповідно список операцій і атрибутів класу. Кожен елемент може мати специфікатор доступу, який визначає, хто має право доступу до операцій та атрибутів:

- public (загальний) – операція або атрибут доступний для використання з будь-якої частини програми будь-яким об'єктом системи;

- protected (захищений) – операція або атрибут доступний об'єктам того класу, де вони оголошені, або об’єктам класів-нащадків;

- private (приватний) – операція або атрибут доступний лише об’єктам того класу, де вони визначені.

Така діаграма встановлює візуальний зв'язок між класами та їхніми характеристиками, надаючи зрозуміле представлення структури програмного забезпечення.

Крім того, в процесі моделювання діаграм класів користувач може вільно визначати атрибути, що відображає особливі для нього властивості чи характеристики класу. Це розширює можливості користувача при створенні моделі, надаючи гнучкість у визначенні та описі об'єктів.

В термінології діаграм класів, під операцією розуміється сервіс, який може виконувати екземпляр класу у відповідь на відповідний виклик. Кожна операція має унікальну назву та список аргументів, які визначаються при виклику. Зазначається також можливість задати тип значення, яке операція повертає, що додає до її опису деталізацію та чіткість. Ця концепція дозволяє точно визначити функціональні можливості кожного класу в системі та його взаємодію з іншими елементами.

Додаткові зв'язки та відношення між класами надають діаграмі класів більш повний обсяг уявлення про взаємодію компонентів системи. Розглянемо деякі з них.

Асоціація -це взаємна залежність між об'єктами різних класів, де кожен клас є рівноправним учасником асоціації. Таке відношення може вказувати на кількість екземплярів об'єктів кожного класу, що беруть участь у даній асоціації (0 - якщо жодного, 1 - якщо один, N - якщо багато).

Залежність -це відношення між класами, де клас-клієнт може використовувати певну операцію або сервіс іншого класу. Зв'язок може бути виражений трасуванням, якщо один клас трансформується в інший внаслідок виконання певного процесу життєвого циклу.

Екземпляризація - це залежність між параметризованим абстрактним класом-шаблоном і реальним класом, який ініціює параметри шаблону. Наприклад, у мові програмування C++, контейнерні класи можуть бути екземпляризовані для різних типів даних.

Композиція - це відношення, схоже на асоціативне, але відмінне тим, що створення об'єкту класу, який використовується, виконується безпосередньо в першому класі. Це вказує на тісний зв'язок між об'єктами, де один об'єкт є частиною іншого.

Генералізація - це відношення розширення одного класу іншим, що вказує на наслідування. Один клас (підклас) успадковує атрибути і методи іншого класу (батьківського класу).

Реалізація - це відношення, де клас реалізує методи, які визначені в інтерфейсі. Клас, який реалізує інтерфейс, повинен надати конкретну реалізацію всіх методів, визначених у відповідному інтерфейсі.

Елементи програми розпізнавання символів систематизовані за допомогою окремих модулів, які взаємодіють між собою, забезпечуючи повний функціонал системи. На рисунку 2.6 наведено UML-діаграму, що відображає взаємодію всіх пакетів програмної системи.

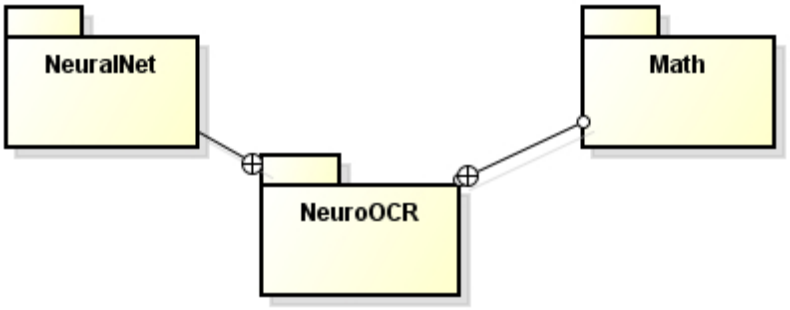

Рисунок 2.6 – UML-діаграма пакетів інформаційної технології оптичного розпізнавання тексту

Детально опишемо ці пакети класів.

Пакет NeuroOCR містить класи, які відповідають за візуальний інтерфейс нашої системи.

Пакет NeuralNet містить класи, які відповідають за основні логічні об’єкти: нейронні мережі, нейрони, шари, навчання нейронних мереж.

Пакет Math містить класи, які відповідають за математичні розрахунки, що потрібні для навчання нейронної мережі.

Проведемо детальний опис класів та інтерфейсів, що входять до цих пакетів.

Пакет: **NeuroOCR**.

Клас: ***MainForm***

Відповідальність: відображення головного вікна програми розпізнавання символів.

Методи:

private void GenerateReceptors() – генерація сенсорів.

private void ReportProgress(int step, string message) – відображення результатів навчання нейронної мережі.

private void GenerateLearningData() – генерація навчальної вибірки.

private void FilterLearningData() – фільтрування навчальної вибірки.

private void CreateNetwork() – створення нейронної мережі.

private void TrainNetwork() – навчання нейронної мережі.

Клас: ***Receptors***

Відповідальність: генерація та збереження сенсорів.

Методи:

public void Add(Receptor receptor) – додати новий сенсор в набір.

public void Generate(int count) – генерація нових сенсорів.

public int[] GetReceptorsState(Bitmap image) – повернення стану сенсора.

Пакет: **NeuralNet**.

Клас: ***BackPropagationLearning***

Відповідальність: навчання із зворотнім поширенням похибки.

Методи:

public float LearnEpoch(float[][] input, float[][] output) – здійснення навчання на одній епосі та повернення сумарної похибки.

public float Learn(float[] input, float[] output) – здійснення однієї ітерації навчання та повернення похибки мережі.

private float CalculateError(float[] desiredOutput) – обчислення похибки нейронної мережі.

private void CalculateUpdates(float[] input) – обчислення значень синапсів нейронної мережі.

private void UpdateNetwork() – встановлення нових ваг у нейронній мережі.

Клас: **ActivationFunction**

Відповідальність: реалізація декількох функцій активації.

Методи:

SigmoidFunction() – реалізація сигмоїдної функції активації.

BipolarSigmoidFunction() – реалізація біполярної сигмоїдної функції активації.

HyperbolicTangensFunction() – реалізація функції активації гіперболічного тангенса.

Клас: **Layer**

Відповідальність: реалізація шару нейронної мережі.

Методи:

public float[] Compute(float[] input) – повертає вихідне значення шару.

Клас: **Network**

Відповідальність: реалізація нейронної мережі.

Методи:

public float[] Compute(float[] input) – повертає вихідне значення нейронної мережі.

Клас: **Neuron**

Відповідальність: реалізація одного нейрона.

Методи:

public float[] Compute(float[] input) – повертає вихідне значення нейрона.

public void Randomize() – встановлення випадкових ваг нейронів.

Пакет: **Math**.

Клас: ***FourierTransform***

Відповідальність: реалізація перетворення Фур'є.

Методи:

public static void DFT(Complex[] data, FourierDirection direction) – реалізація одновимірного дискретного перетворення Фур'є.

public static void DFT2(Complex[,] data, FourierDirection direction) – реалізація двовимірного дискретного перетворення Фур'є.

public static void FFT(Complex[] data, FourierDirection direction) – реалізація одновимірного швидкого перетворення Фур'є.

public static void FFT2(Complex[,] data, FourierDirection direction) - реалізація двовимірного швидкого перетворення Фур'є.

Клас: ***Tools***

Відповідальність: реалізація деяких математичних дій.

Методи:

public static int Pow2(int exp) – реалізація піднесення в квадрат.

public static int Log2(int x) – реалізація логарифму з основою 2.

Загальна UML-діаграма класів програма наведена на рисунку 2.7.

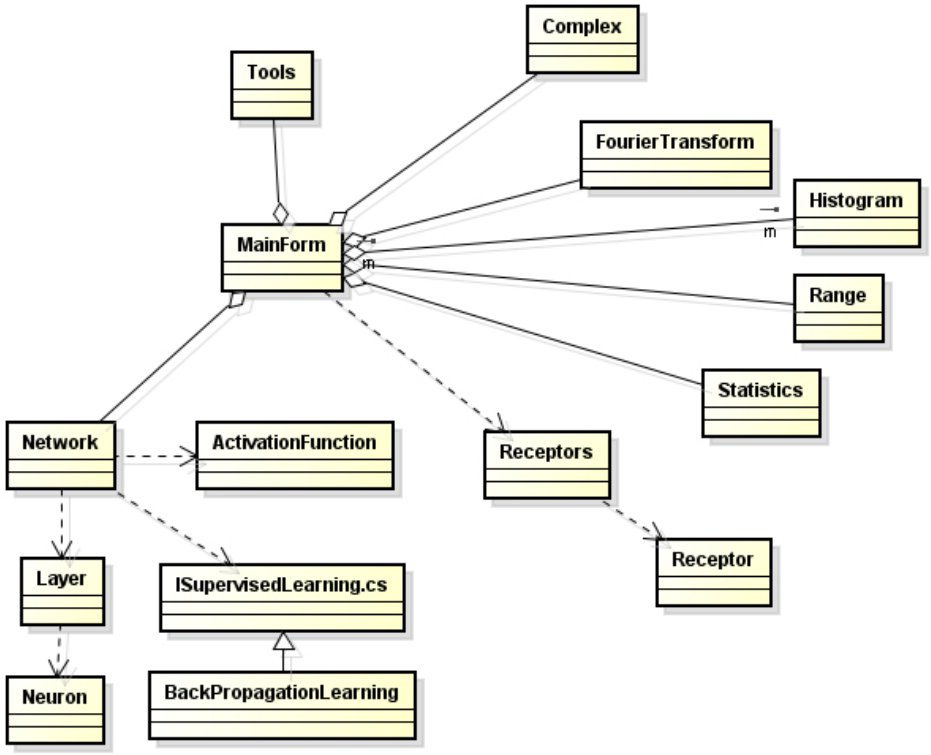

Рисунок 2.7 – UML-діаграма класів програми розпізнавання тексту

На основі цієї діаграми було здійснено програмну реалізацію інформаційної технології розпізнавання тексту на базі нейронної мережі багатошаровий персептрон.

2.6 Висновок до розділу 2

В даному розділі магістерської кваліфікаційної роботи було досліджено і розроблено основні етапи, з яких складається процес розпізнавання тексту. Було обґрунтовано вибір типу нейронної мережі, розглянуто архітектуру та математичну модель нейронної мережі багатошаровий персептрон, яка навчається за алгоритмом зворотного поширення помилки. Проаналізовано роботу нейронної мережі багатошаровий персептрон в режимі розпізнавання символів. У підсумку було розроблено структуру інформаційної технології оптичного розпізнавання тексту, розроблено алгоритм роботи та UML-діаграму класів програмного забезпечення розпізнавання тексту.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ОПТИЧНОГО РОЗПІЗНАВАННЯ ТЕКСТУ

3.1 Обґрунтування вибору мови та середовища програмування

Мова програмування C# представляє собою об'єктно-орієнтовану мову з безпечною системою типізації, спеціально призначену для використання на платформі .NET. Її розробка відбувалася в рамках дослідницького підрозділу Microsoft Research, зусиллями видатних фахівців, таких як Андерс Гейлсберг, Скот Вілтамут і Пітер Гольде.

C# створювалася з метою надання мови програмування високого рівня для використання на платформі CLR (Common Language Runtime). Завдяки цьому, мова тісно взаємодіє з функціоналом CLR, що визначає систему типізації та інші характеристики C#. У розвитку мови протягом версій від 1.1 до 2.0 спостерігалося значне збагачення її можливостей, обумовлене саме розвитком CLR. Зазначимо, що версія C# 3.0 порушила цю закономірність, додаючи розширення мови, які не базуються на розширеннях платформи .NET.

Однією з основних переваг C# є використання CLR, яка надає мові та іншим .NET-орієнтованим мовам ряд можливостей, недоступних для "класичних" мов програмування. Наприклад, збірка сміття в C# не реалізована безпосередньо в мові, а здійснюється CLR для програм, написаних на C# [13].

C# – це мова програмування, яка зосереджується на об’єктах і даних. Вона слідує принципам об’єктно-орієнтованого програмування (ООП), що робить її ідеальним інструментом для опису та моделювання складних систем.

У C# дані та функції, які з ними пов’язані, упаковуються в об’єкти, що сприяє більш логічному і структурованому коду. Типобезпека – ще одна важлива характеристика C#. Це означає, що під час роботи з даними в C# ви зобов’язані явно вказувати їхній тип, що зменшує можливість помилок і робить ваш код надійним.

C# тісно інтегрована з .NET (Microsoft .NET Framework) – потужною платформою для розробки та виконання додатків. Важливою частиною .NET є CLR

(Common Language Runtime), яка забезпечує середовище виконання для C# та інших мов .NET. CLR забезпечує керування пам’яттю, безпеку та інші важливі функції, що дає змогу C# розробникам зосереджуватися на створенні коду, а не на низькорівневих деталях.

Інтеграція з .NET також означає доступ до великої бібліотеки класів, яка містить готові рішення для безлічі завдань. Це скорочує час розробки і дозволяє створювати більш функціональні додатки.

Ця мова програмування підтримує безліч різних платформ, включно з Windows, macOS і Linux. Це означає, що ви можете створювати додатки, які працюють практично скрізь.

Завдяки інструментам, таким як .NET Core і .NET 5 (та їхнім наступним версіям), C# розробники можуть створювати крос-платформні додатки, які не залежать від конкретної операційної системи. Це особливо важливо в сучасному світі, де різноманітність пристроїв і платформ перебуває на піку.

Принципові рішення, впроваджені у мові програмування C#, визначають її основні характеристики та функціональні можливості:

- компонентно-орієнтований підхід до програмування. Цей принцип базується на ідеї розробки програм за допомогою компонентів, які можна знову використовувати. У мові C#, це реалізується через використання поняття класів та об'єктно-орієнтованого програмування (ООП);

- властивості як засіб інкапсуляції даних. Властивості дозволяють забезпечити контроль доступу до полів класів та забезпечити шлях для читання та запису їх значень, зберігаючи при цьому інкапсуляцію даних;

- обробка подій. Мова C# надає розширені засоби для роботи з подіями, включаючи обробку винятків за допомогою оператора try;

- делегати у C# дозволяють вказувати на функції або методи як параметри, реалізуючи тим самим покажчик на функцію в мовах C і C++.

- індексатори дозволяють звертатися до елементів класу-контейнера за допомогою операторів індексу, схожих на роботу з масивами;

- перевантажені оператори. В C# можна перевантажувати оператори, щоб визначити власне значення для використання їх з екземплярами класів;

- оператор foreach. Цей оператор дозволяє легко перебирати всі елементи класів-колекцій, спрощуючи обробку даних;

- механізми boxing і unboxing. Boxing дозволяє конвертувати значимі типи даних в типи-об'єкти, а unboxing – навпаки;

- атрибути використовуються для додавання метаданих до коду, що спрощує взаємодію з COM-моделлю та іншими аспектами програмування;

- прямокутні масиви визначаються як набір елементів з доступом за номером індексу, і вони мають однакову кількість стовпців і рядків.

Ці основні рішення формують основу C# як сучасної та потужної мови програмування для платформи .NET.

C# активно використовується у сфері веб-розробки. Одним із ключових інструментів для створення веб-додатків на C# є ASP.NET – платформа, що надає розробникам потужні засоби для побудови сучасних і продуктивних веб-додатків. ASP.NET пропонує широкий набір інструментів для роботи з веб-технологіями, базами даних, безпекою та іншими аспектами веб-розробки. C# і ASP.NET дають змогу створювати динамічні та масштабовані веб-додатки, обробляти запити від користувачів і взаємодіяти з базами даних. Це означає, що ви можете створювати різноманітні веб-сайти, від корпоративних порталів до електронних магазинів.

C# має тісну інтеграцію з операційною системою Windows, що робить його чудовим вибором для створення настільних додатків під цю платформу. За допомогою технології Windows Forms або більш сучасної Universal Windows Platform (UWP), ви можете розробляти додатки з графічним інтерфейсом користувача (GUI), які інтегровані з операційною системою Windows.

C# також популярна серед розробників ігор, особливо завдяки ігровому рушію Unity. Це потужний засіб для створення 2D і 3D ігор, а також віртуальної та доповненої реальності. Одна з його ключових переваг — підтримка C# для програмування ігрової логіки. Використовуючи C# і Unity, розробники можуть створювати ігри для різних платформ, включно з ПК, консолями, мобільними

пристроями і навіть віртуальними шоломами. Unity надає багату бібліотеку ресурсів, інструменти для моделювання та анімації об’єктів, а також підтримує фізичний рушій для створення реалістичної поведінки об’єктів у грі.

Ці три галузі — лише невеликий огляд того, де використовується мова сі шарп. Можливості цієї мови розширюються щодня, і її гнучкість робить її універсальним інструментом для безлічі різних проєктів.

Microsoft Visual Studio - це комплексний набір інструментів для розробки програмного забезпечення, розроблений фірмою Microsoft. Основною метою Visual Studio є надання інтегрованого середовища розробки (IDE) та ряду інших засобів для зручного створення, тестування та вдосконалення програм. Детальніше про деякі з ключових характеристик Visual Studio:

- інтегроване середовище розробки (IDE). Visual Studio надає потужне та зручне IDE, яке об'єднує в собі редактор коду, відладчик, дизайнер інтерфейсів, інструменти для збірки та управління версіями;

- підтримка різноманітних технологій. Visual Studio дозволяє розробляти різноманітні типи програм, включаючи консольні програми, застосунки з графічним інтерфейсом (включаючи Windows Forms), веб-сайти, веб-застосунки та служби, як в рідному, так і в керованому кодах;

- підтримка платформ Microsoft. Visual Studio орієнтований на розробку для різних платформ Microsoft, таких як Microsoft Windows, Windows Mobile, Windows Phone, Windows CE. Він також підтримує різні версії .NET Framework, .NET Compact Framework і Microsoft Silverlight;

- мови програмування. Visual Studio підтримує різні мови програмування, включаючи C#, Visual Basic, C++, F# та інші. Кожна мова має своє відповідне середовище та інструменти;

- тестування та відлагодження. Інтегровані інструменти для автоматичного та ручного тестування, а також відлагодження дозволяють розробникам створювати високоякісне програмне забезпечення;

- спільна робота та керування кодом. Visual Studio забезпечує інструменти для спільної роботи команд розробників, а також можливості керування версіями коду за допомогою системи контролю версій;

- розширення та розширюваність. Велика кількість додаткових компонентів, шаблонів та розширень дозволяє розробникам розширювати можливості Visual Studio та пристосовувати її до своїх потреб.

Visual Studio є ключовим інструментом для розробки на платформі Microsoft, і його функціональність охоплює весь життєвий цикл розробки програмного забезпечення.

Visual Studio – це невід'ємна частина робочого процесу для багатьох розробників, які спеціалізуються на платформі Microsoft. Цей інтегрований розробницький середовище (IDE) надає широкий спектр інструментів і можливостей, які допомагають спростити і поліпшити кожен етап створення програмного забезпечення.

Починаючи з створення нового проекту, Visual Studio пропонує різноманітні шаблони, які включають в себе різні типи додатків, від класичних консольних програм до складних веб-додатків і мобільних застосунків. Інтуїтивний інтерфейс сприяє швидкому старту, а інтегровані засоби автоматизації дозволяють зосередитися на креативному процесі розробки.

Під час активного кодування розробники можуть скористатися багатьма корисними функціями, такими як автоматичне доповнення коду, перевірка синтаксису, а також вбудовані інструменти для відлагодження та профілювання програми. Завдяки інтеграції з системами контролю версій, розробники можуть ефективно керувати версіями свого коду та спілкуватися з іншими членами команди.

Інструменти Visual Studio відображають сучасні вимоги розробки програмного забезпечення, допомагаючи не лише в створенні традиційних додатків для мобільних телефонів і комп'ютерів, але й у розробці хмарних застосунків. Процеси тестування, відлагодження і розгортання програм в хмарі аналогічні процесам створення .NET-застосунків. Важливою новацією в Visual Studio є інструменти для багатопоточної розробки, які охоплюють як некерований, так і .NET Framework.

Visual Studio виводить на новий рівень інтерфейс, використовуючи Windows Presentation Foundation (WPF). Уведено наступне покоління інструментів ASP.NET, забезпечено підтримку динамічних розширень в мовах програмування C# і Visual Basic. Також представлені нові шаблони проектів, інструментарій для документування тестових сценаріїв та значна кількість нових бібліотек, спрямованих на підтримку функцій Windows 7.

Visual Studio Ultimate, що формально відома як Visual Studio Team System з кодовою назвою Rosario, представляє собою новий інструмент для спільної розробки застосунків. Інтегроване середовище розробки (IDE - Integrated Development Environment) Visual Studio є багатовіконним та володіє широким спектром можливостей. Усередині його можна виділити три основні вікна:

- вікно «Обозревателя решений» (Solution Explorer). В цьому вікні представлена структура побудованого рішення, що дозволяє оглядати та керувати різними елементами проекту;

- вікно властивостей (Properties). У вікні Свойства можна переглядати та редагувати властивості вибраного елемента рішення. Це дозволяє змінювати параметри та налаштування обраних об'єктів у проекті;

- вікно документів. В цьому вікні відображається вміст вибраного документа або файлу. Це вікно також може відображати інші документи, а список доступних документів показується у верхній частині вікна.

Visual Studio Ultimate надає зручний та ефективний інтерфейс для розробки та спільної роботи над проектами.

На рисунку 3.1 представлено зовнішній вигляд головного вікна Visual Studio.

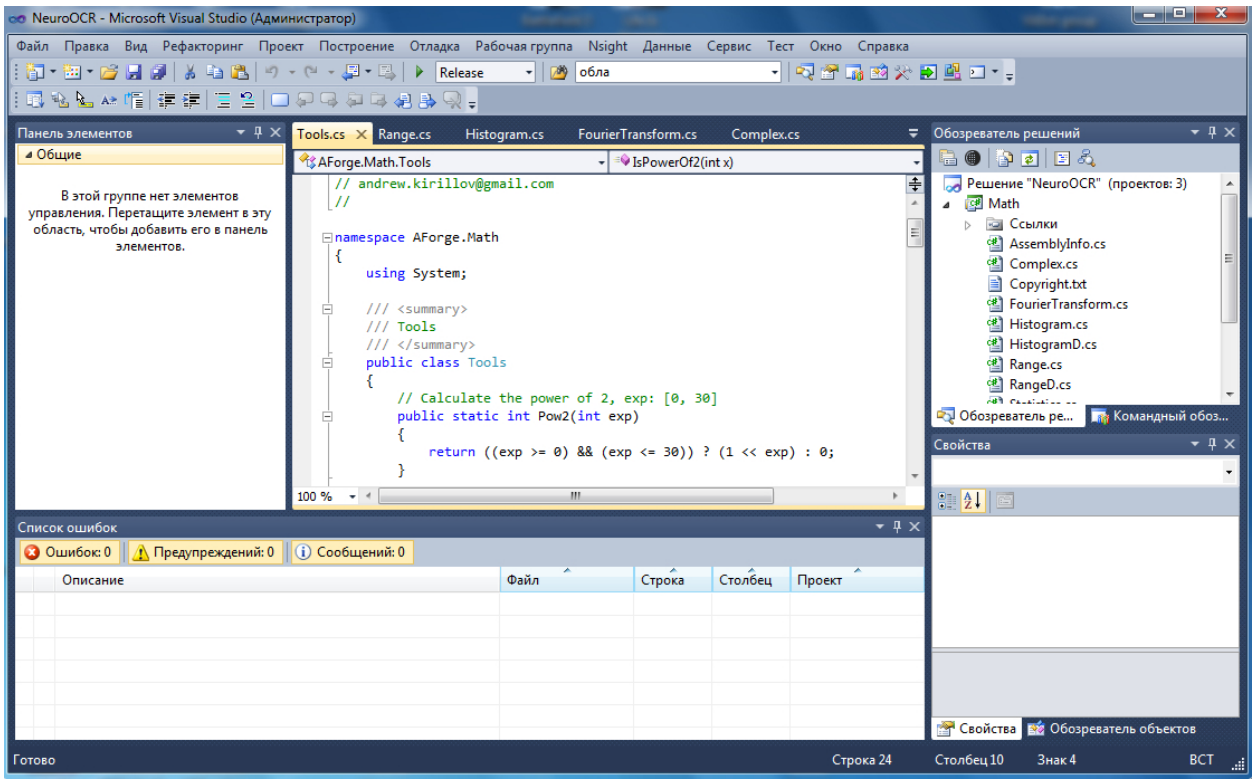

Рисунок 3.1 – Головне вікно середовища програмування Visual Studio

3.2 Реалізація головного вікна програмного забезпечення розпізнавання тексту

Windows Forms представляє собою прикладний програмний інтерфейс (API), який відповідає за графічний інтерфейс користувача та є невід'ємною частиною Microsoft .NET Framework. Цей інтерфейс спрощує доступ до елементів інтерфейсу Microsoft Windows, створюючи обгортку для існуючого Win32 API в керованому коді. Керований код, який реалізує API для Windows Forms, не залежить від мови розробки, що робить його універсальним для використання в різних мовах програмування, таких як C#, C++, VB.NET, J#, та інші [16].

Основні особливості Windows Forms включають можливість легко створювати графічний інтерфейс користувача для Windows додатків, використовуючи візуальні елементи, такі як кнопки, тексти, списки тощо. Інтерфейс також надає можливість обробки подій, таких як натискання кнопок або введення тексту.

Windows Forms забезпечує розробникам гнучкість у виборі мови програмування і використанні API для створення відмінних застосунків з графічним інтерфейсом для операційних систем Windows.

Windows Forms може розглядатися як еволюційна зміна старішої і важкоосвоєної бібліотеки MFC, яка спочатку була написана мовою C++. Однак Windows Forms не надає парадигму, яка б була аналогічною до MVC (Model-View-Controller). З метою вирішення цієї ситуації та реалізації необхідної функціональності для Windows Forms існують сторонні бібліотеки.

Однією з таких широко використовуваних бібліотек є User Interface Process Application Block (UIPAB), яку розробила спеціальна група від Microsoft, що відповідає за приклади і рекомендації, і доступна для безкоштовного завантаження. UIPAB дозволяє впроваджувати парадигму Model-View-Controller в середовище Windows Forms, забезпечуючи більшу гнучкість у відділенні логіки програми від представлення і управління взаємодією з користувачем. Крім того, бібліотека містить вихідний код і навчальні приклади, які сприяють прискоренню навчання та впровадженню нових концепцій.

Отже, за допомогою Windows Forms було реалізовано графічний інтерфейс програмого забезпечення оптичного розпізнавання тексту.

На рисунку 3.2 зображено головне вікно програми розпізнавання тексту, яке створене в класі MainForm. В цьому вікні реалізовані область для малювання символу, блок генерації сенсорів, блок генерації навчальної вибірки, вибору шрифтів, блок генерації даних, блок створення нейронної мережі із вибором кількості шарів та параметрів навчання: кількість епох, швидкість навчання на кожному проході, поріг похибки, блок із результатами навчання: сумарна похибка, неправильно класифіковані символи, та результат роботи самої мережі у вигляді розпізнаного символу.

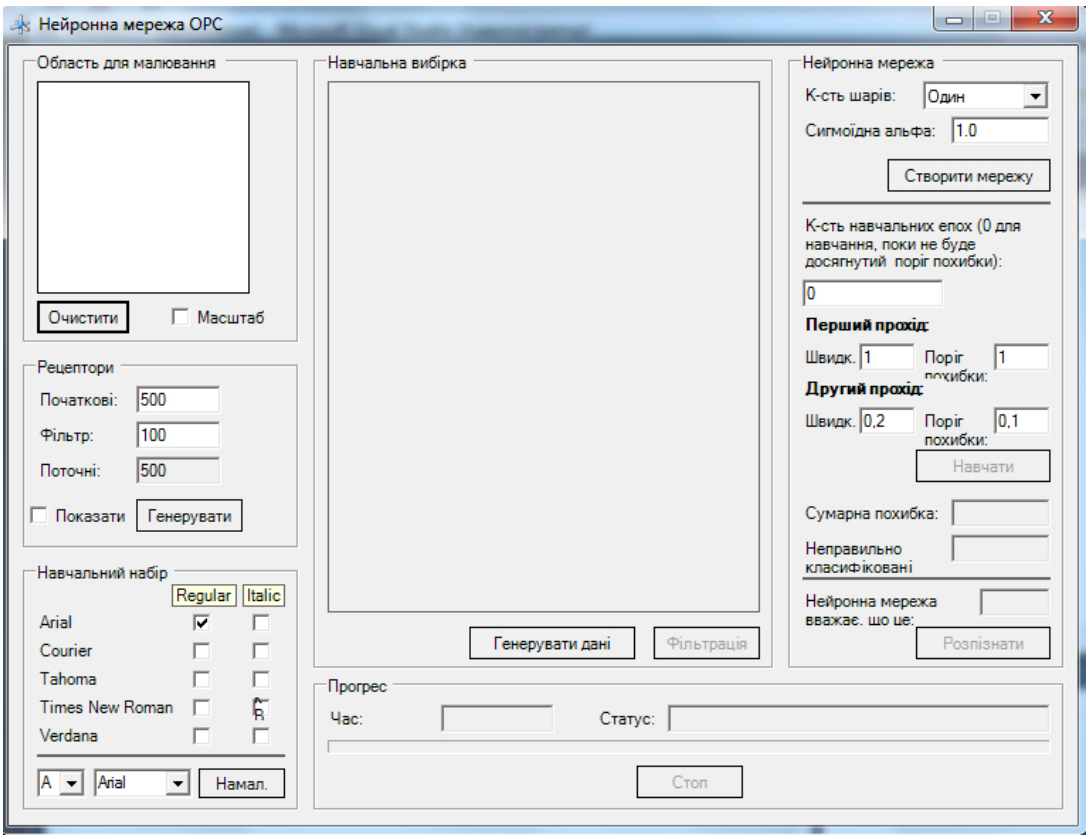

Рисунок 3.2 – Початкове вікно програми розпізнавання символів

3.3 Розробка програмного забезпечення оптичного розпізнавання тексту

Приведемо реалізацію деяких основних методів розробленої програми розпізнавання символів. На рисунку 3.3 наведений метод LearnEpoch () класу BackPropagation Learning, що реалізує одну епоху навчання нейронної мережі.

```
public float LearnEpoch(float[][] input, float[][] output)
{
    int     i, n = input.Length;
    float   error = 0.0f;

    // for all training patterns
    for (i = 0; i < n; i++)
    {
        error += Learn(input[i], output[i]);
    }
    // determine if we converged
    converged = (error < learningLimit);

    // return error
    return error;
}
```

Рисунок 3.3 - Метод LearnEpoch () класу BackPropagation

На рисунку 3.4 наведений метод Learn() класу BackPropagation Learning, що реалізує одну ітерацію навчання нейронної мережі.

```
public float Learn(float[] input, float[] output)
{
    // compute the network
    float[] nout = net.Compute(input);

    // calculate network error
    float error = CalculateError(output);

    // calculate weights updates
    CalculateUpdates(input);

    // update the network
    UpdateNetwork();

    // return error level
    return error;
}
```

Рисунок 3.4 - Метод Learn () класу BackPropagation

На рисунку 3.5 наведений метод CalculateError() класу BackPropagation Learning, що реалізує обчислення похибки мережі.

```
// calculate error for the last layer
layer = net[layersCount - 1];
err = errors[layersCount - 1];

for (i = 0, n = layer.NeuronsCount; i < n; i++)
{
    output = layer[i].Output;
    // error of the neuron
    e = desiredOutput[i] - output;
    // error multiplied with first derivative
    err[i] = e * function.OutputPrime2(output);
    // squre the error and sum it
    error += (e * e);
}

// calculate error for other layers
for (j = layersCount - 2; j >= 0; j--)
{
    layer       = net[j];
    layerNext   = net[j + 1];
    err         = errors[j];
    errNext     = errors[j + 1];

    // for all neurons of the layer
    for (i = 0, n = layer.NeuronsCount; i < n; i++)
    {
        sum = 0.0f;
        // for all neurons of the next layer
        for (k = 0, m = layerNext.NeuronsCount; k < m; k++)
        {
            sum += errNext[k] * layerNext[k][i];
        }
        err[i] = sum * function.OutputPrime2(layer[i].Output);
    }
}
```

Рисунок 3.5 - Метод CalculateError() класу BackPropagation

3.4 Тестування та аналіз результатів роботи програмного забезпечення оптичного розпізнавання тексту

Запустимо та протестуємо розроблену програму розпізнавання тексту. Для цього запустимо файл NewOCR.exe.

Далі потрібно в блоці «Навчальний набір» відмітити галочками усі типи шрифтів, які наша нейронна мережа буде навчатись розпізнавати. Серед доступних в програмі шрифтів є такі: Arial, Courier, Tahoma, Times New Roman, Verdana. Для кожного з шрифтів можна вибрати його тип: стандартний або курсивний. Вибір шрифтів зображений на рисунку 3.6.

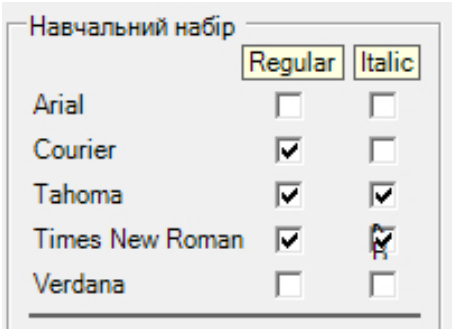

Рисунок 3.6 – Блок програми, що відповідає за навчальний набір шрифтів

Потім необхідно обрати кількість сенсорів, що будуть відповідати за якість навчання та розпізнавання символів. За замовчуванням, кількість початкових сенсорів дорівнює 500. Також потрібно обрати кількість сенсорів, які залишаться після їх фільтрації. За замовчуванням, їх кількість рівна 100. Потім потрібно натиснути кнопку «Генерувати» і програма згенерує випадковим чином набір сенсорів. Побачити їх в області для малювання можна, відмітивши галочку «Показати». Згенеровані програмою сенсори зображені на рисунку 3.7.

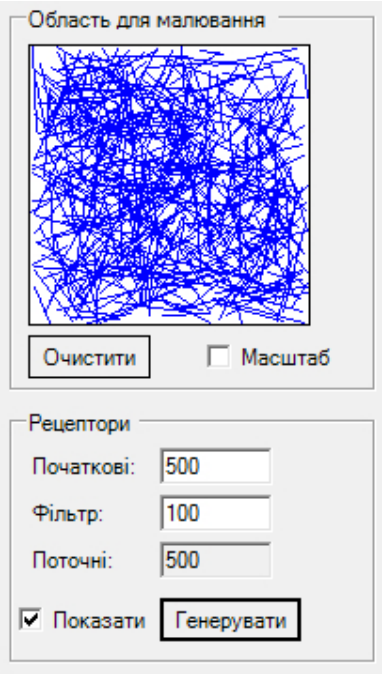

Рисунок 3.7 – Згенеровані програмою розпізнавання тексту сенсори

Далі потрібно натиснути кнопку «Генерувати дані», щоб програма сформувала набір даних для навчання нейронної мережі. Кількість навчальних даних можна зменшити відібравши лише якісні сенсори. Для цього потрібно натиснути кнопку «Фільтрація».

Далі в блоці «Нейронна мережа» необхідно вибрати кількість шарів нейронної мережі (в програмі доступні 1 або 2 шари) та значення параметра «альфа» сигмоїдної функції активації, і натиснути кнопку «Створити мережу».

Потім потрібно вказати кількість епох, протягом яких буде проводитись навчання або залишити 0 для навчання, поки не буде досягнутий поріг похибки. Нижче вказати швидкість першого та другого проходу навчання, а також значення похибки, при якій навчання буде зупинятись. Потім натиснути кнопку «Навчати» та очікувати поки нейронна мережа не закінчить навчання. Один з прикладів налаштування параметрів для навчання нейронної мережі зображено на рисунку 3.8.

Процес навчання зображено на рисунку 3.9. Навчання можна зупинити в будь-який момент, натиснувши кнопку «Стоп». В результаті отримуємо сумарну похибку та кількість неправильно класифікованих символів.

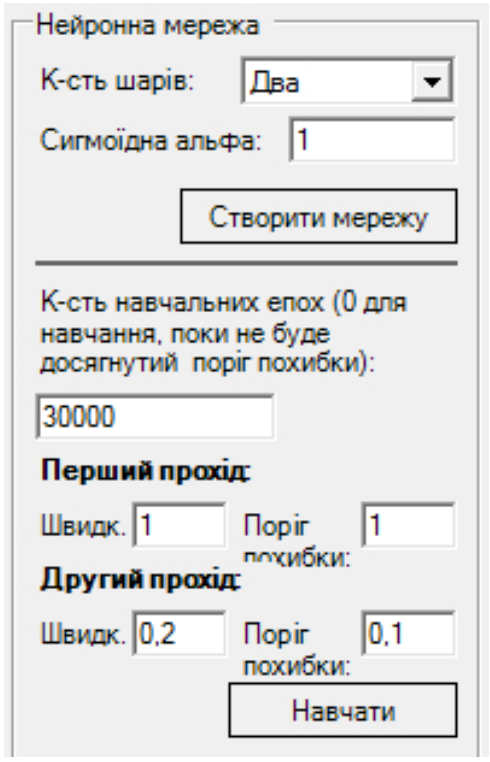

Рисунок 3.8 – Налаштування параметрів навчання нейронної мережі

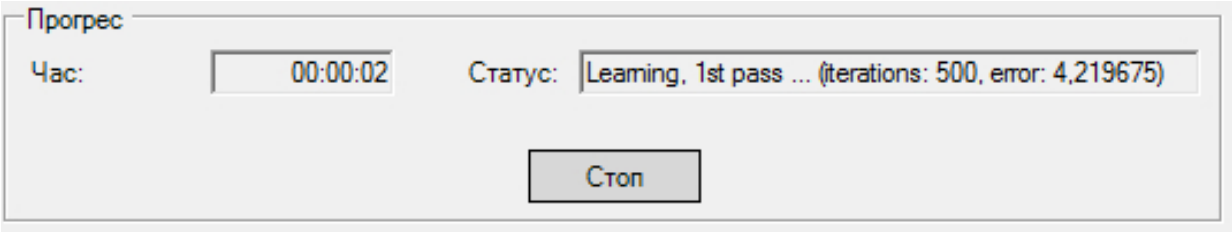

Рисунок 3.9 – Процес навчання нейронної мережі

Після навчання потрібно обрати символ для розпізнавання. Це можна зробити двома шляхами: намалювати вручну в області для малювання або вибрати шрифт та символ і програма намалює автоматично. І натиснути кнопку «Розпізнати», після чого мережа виводить результат. Приклад вхідних та вихідних даних розпізнавання символу зображено на рисунках 3.10 і 3.11 відповідно.

Рисунок 3.10 – Намальований вручну символ для розпізнавання

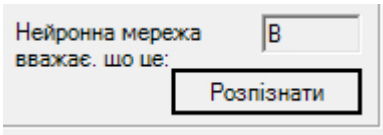

Рисунок 3.11 – Результат розпізнавання програмою намальованого символу

Загальний вид вікна з результатом роботи розробленої програми розпізнавання символу F зображено на рисунку 3.12

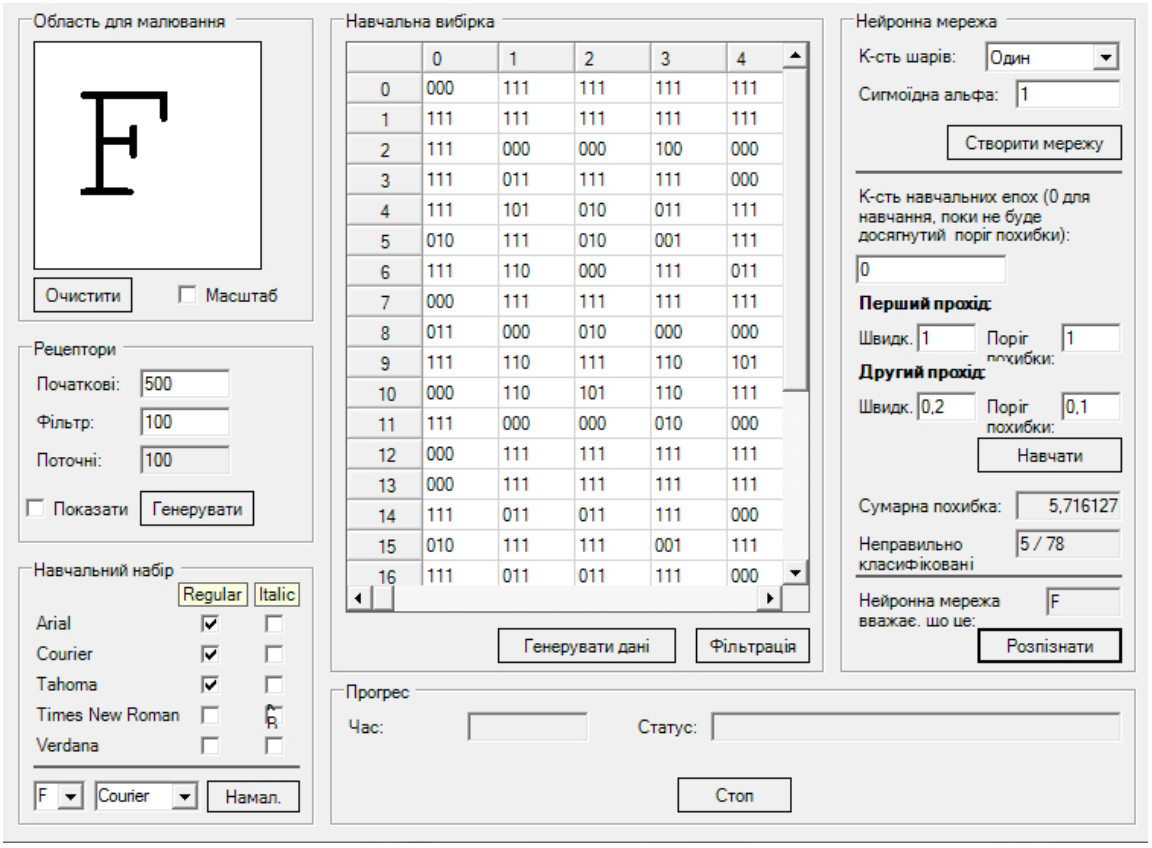

Рисунок 3.12 – Загальний вид вікна з результатом розпізнавання символу F

Для тестування програми розпізнавання символів виберемо таку вибірку:

- шрифт Arial Regular, літери D, G, M, Z;
- шрифт Verdana Regular, літери A, F, P, S;
- шрифт Times New Roman Italic, літери E, J, Q, W;
- літери B, H, N намальовані вручну.

Проаналізуємо, як результати роботи програми розпізнавання символів залежать від кількості шарів у нейронній мережі. Для цього тесту задамо такі параметри:

- шрифти: «Arial Regular», «Courier Regular», «Tahoma Regular»
- кількість епох: 30000
- швидкість навчання на першому проході: 1
- швидкість навчання на другому проході: 0,2

Розпізнавання за допомогою нейронної мережі з одним шаром нейронів показало хороші результати на Regular-шрифтах, але мережа не змогла розпізнати курсивний шрифт, а також намальовані рукою літери. Однак з цим досить добре справилась нейронна мережа з двома шарами нейронів. Повні результати цього тестування наведено у таблиці 3.1.

Таблиця 3.1 – Результати тестування розробленої програми, що показують залежність розпізнавання від кількості шарів в нейронній мережі

К-сть шарів	К-сть епох	Сумарна похибка	Неправильно класифіковані	Час навчання, с	Розпізнаних символів з тест. вибірки	Достовірність розпізнавання, %
1	30000	9,116	8/78	96	9/15	60
2	30000	1,418	1/78	123	13/15	86,6

Далі проаналізуємо, як результати роботи програми розпізнавання символів залежать від швидкості навчання нейронної мережі. Встановимо один шар нейронів в мережі. Результати показали, що при меншій швидкості навчання, нейронна мережа краще розпізнає символи. На рисунку 3.13 зображено результат розпізнавання вручну

намальованої літери N. Як бачимо, при високій швидкості навчання та одному шарі нейронів, наша нейронна мережа не справляється з розпізнавання вручну намальованих символів. Повні результати цього тестування наведено у таблиці 3.2.

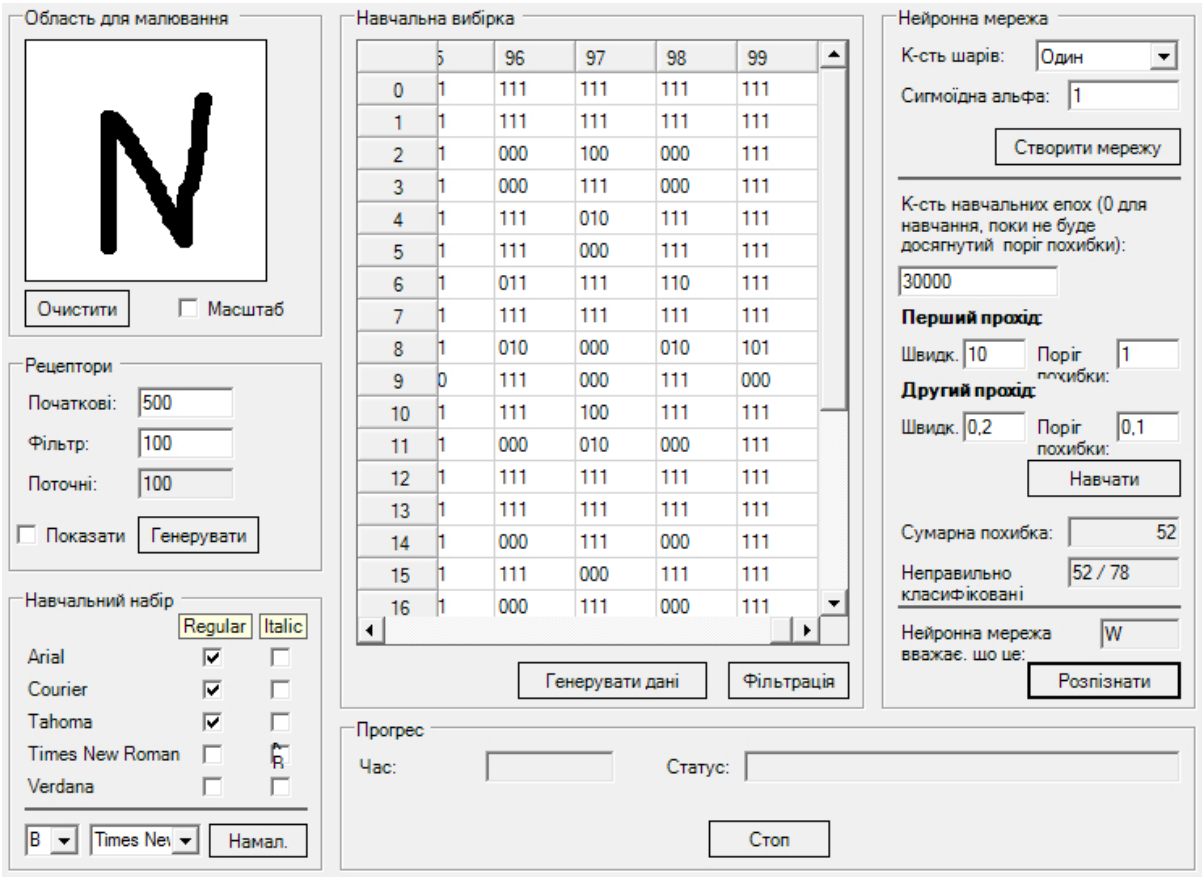

Рисунок 3.13 – Результат розпізнавання намальованої вручну літери N

Таблиця 3.2 – Результати тестування розробленої програми, що показують залежність розпізнавання від швидкості навчання нейронної мережі

Швидкість навчання на першому проході	К-сть епох	Сумарна похибка	Неправильно класифіковані	Час навчання, с	Розпізнаних символів з тест. вибірки	Достовірність розпізнавання, %
1	30000	5,857	4/78	71	10/15	66,7
10	30000	56,021	52/78	85	5/15	33,3

Проаналізуємо також, як результати роботи програми розпізнавання символів залежать від кількості сенсорів нашої нейронної мережі. Встановимо два шари нейронів в мережі. Результати показали, що при більшій кількості сенсорів, нейронна мережа краще розпізнає символи. Проте, на навчання мережі потрібно затрачати більше часу. Також результати розпізнавання залежать від того як були згенеровані сенсори. Повні результати цього тестування наведено у таблиці 3.3.

Таблиця 3.3 – Результати тестування розробленої програми, що показують залежність розпізнавання від кількості сенсорів нейронної мережі

Кількість сенсорів	К-сть епох	Сумарна похибка	Неправильно класифіковані	Час навчання, с	Розпізнаних символів з тест. вибірки	Достовірність розпізнавання, %
25	30000	26,56	22/78	56	5/15	33,3
100	30000	9,04	7/78	124	7/15	46,7
500	30000	8,95	8/78	182	8/15	53,3

Отже, протестувавши розроблену програму розпізнавання тексту можна побачити, що програма здійснює розпізнавання досить добре, особливо це залежить від правильного налаштування параметрів нейронної мережі. Краще розпізнавання буде здійснюватись при більшій кількості шарів в нейронній мережі, при меншій швидкості навчання, а також при більшій кількості сенсорів. Проте буде збільшуватись кількість часу, що затрачується на навчання. Тому потрібно шукати певний баланс між часом навчання та параметрами мережі. Найкраще розпізнавання розроблена програма здійснює на regular-шрифтах. Гірші результати було отримано на курсивних шрифтах та намальованих вручну символах.

Для доведення досягнення поставленої в роботі мети - підвищення достовірності розпізнавання символів – було протестовано роботу розробленої програми та

програми-аналога CuneiForm [6] на одних і тих самих шрифтах. Результати цього тестування представлені у таблиці 3.4.

Таблиця 3.4 – Результати тестування розробленої програми та програми-аналога CuneiForm

Програмний засіб	К-сть символів у тест. вибірці	Неправильно класифіковані	Достовірність розпізнавання, %
Програма CuneiForm	5200	426	91,8
Розроблена програма	5200	171	96,7

Із табл. 3.4 видно, що розроблена програма має вищу достовірність розпізнавання (96,7%), ніж аналогічна програма (91,8%), а значить мета роботи досягнута.

3.5 Висновок до розділу 3

В ході практичної реалізації інформаційної технології оптичного розпізнавання тексту розглянуто та обґрунтовано переваги та недоліки об’єктно-орієнтованої мови програмування високого рівня C# та середовища Microsoft Visual Studio. В результаті було спроектовано власне програмне забезпечення розпізнавання тексту. Програму написано об’єктно-орієнтованою мовою програмування високого рівня C# з використанням Windows Forms під операційну систему Windows. Програма має віконний інтерфейс. Користувач може обирати тип, вид написання, розмір та колір шрифта, який береться як еталонний. Також може малювати зображення символу, який підлягає розпізнаванню, розпізнавати намальоване зображення символу за допомогою нейронної мережі. В результаті даної роботи отримано програмне забезпечення розпізнавання тексту на основі нейронної мережі багатошаровий персептрон, який порівняно з аналогом має кращу на 4,9% достовірність розпізнавання текстових символів (96,7%), ніж аналогічна програма (91,8%).

4 ЕКОНОМІЧНА ЧАСТИНА

Ефективне впровадження науково-технічної розробки стає можливим, якщо вона відповідає поточним вимогам науково-технічного прогресу та враховує економічні аспекти. Надання оцінки економічної ефективності отриманих результатів науково-дослідної роботи є важливою частиною цього процесу.

Магістерська робота, присвячена розробці та дослідженню «Інформаційна технологія оптичного розпізнавання тексту», віднесена до науково-технічних робіт, спрямованих на введення на ринок. Рішення про комерціалізацію розробки може бути прийняте протягом самої роботи, відкриваючи можливість виведення її на ринок. Цей напрямок розглядається як пріоритетний, оскільки розроблені результати можуть бути корисними для різних зацікавлених сторін і приносити економічні вигоди. Проте для успішного втілення цього процесу важливо знайти зацікавленого інвестора, який був би зацікавлений у реалізації цього проекту, і переконати його в обґрунтованості таких інвестицій.

Для цього визначені наступні етапи виконання робіт:

1. Проведено комерційний аудит науково-технічної розробки, що включає в себе визначення науково-технічного рівня та комерційного потенціалу.

2. Розраховані витрати на реалізацію науково-технічної розробки.

3. Проведено розрахунок економічної ефективності науково-технічної розробки в разі її впровадження та комерціалізації потенційним інвестором, а також обґрунтовано економічну доцільність комерціалізації для інвестора.

4.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Інформаційна технологія оптичного розпізнавання тексту» є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями, наведеними в таблиці 4.1 [20].

Таблиця 4.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в	Технічні та споживчі влас-тивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в	Технічні та споживчі властивості продукту значно кращі, ніж в
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає

Продовження таблиці 4.1

Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь- які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки потрібно зведені до таблиці 4.2. Для опитування було залучені експерти: Королюк Д.І. (Lead Systems Engineer, EPAM), Борка М.Ю (Senior Systems Engineer, EPAM), Доманський Б.В. (Senior Systems Engineer, EPAM).

Таблиця 4.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	Королюк Д.І.	Борка М.Ю	Доманський Б.В.
	Бали:		
1. Технічна здійсненність концепції	4	4	4
2. Ринкові переваги (наявність аналогів)	4	4	4
3. Ринкові переваги (ціна продукту)	2	2	2
4. Ринкові переваги (технічні властивості)	4	3	4
5. Ринкові переваги (експлуатаційні витрати)	3	3	3
6. Ринкові перспективи (розмір ринку)	2	3	3
7. Ринкові перспективи (конкуренція)	2	3	3
8. Практична здійсненність (наявність фахівців)	3	3	3
9. Практична здійсненність (наявність фінансів)	3	3	3
10. Практична здійсненність (необхідність нових матеріалів)	3	3	3
11. Практична здійсненність (термін реалізації)	4	4	4
12. Практична здійсненність (розробка документів)	2	2	2
Сума балів	$СБ_1=31$	$СБ_2=34$	$СБ_3=34$
Середньоарифметична сума балів $СБ_c$	$\overline{СБ}=\frac{\sum_1^3 СБ_i}{3}=\frac{31+34+34}{3}=33$		

За результатами розрахунків, наведених в таблиці 4.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в таблиці 4.3 [20].

Таблиця 4.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ , розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою "Інформаційна технологія оптичного розпізнавання тексту" становить 33 бали, що, відповідно до таблиці 4.3, свідчить про комерційну важливість проведення даних досліджень оскільки рівень комерційного потенціалу розробки вище середнього.

Результатом магістерської роботи є процес оптичного розпізнавання тексту. Результати будуть корисні розробникам програмного забезпечення, які можуть інтегрувати данну технологію в будь який продукт, який працює з зображеннями тексту.

Програма розпізнає текст на англійській, болгарській, голландській, данській, естонській, іспанській, італійській, латвійській, литовській, німецькій, польській, португальській, російській, румунській, сербській, словенській, турецькій, угорській, українській, французькій, хорватській, чеській, шведській мовах та російсько-англійський двомовний текст.

CuneiForm може зберегти розпізнаний текст у форматах RTF, HTML, або текстовому. Також можливо передати текст до текстового процесора Word або електронної таблиці Excel.

На сьогоднішній день CuneiForm є системою для перетворення електронних документів, а також зображень в редагований вигляд, дозволяючи при цьому зберігати структуру і шрифти оригіналу, як в автоматичному, так і в напівавтоматичному режимі.

Головним недоліком системи аналогу є невелика точність розпізнавання (90%). Ще одним недоліком даного програмного комплексу є його порівняно велика

вартість, що робить його недоступним для більшості користувачів. Як і програма, що розробляється, даний аналог також написаний під операційну систему Windows.

4.2 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему "Інформаційна технологія оптичного розпізнавання тексту", під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою [20]:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \qquad (4.1)$$

де k – кількість посад дослідників залучених до процесу досліджень;

M_{ni} – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дн.;

T_p – середнє число робочих днів в місяці, T_p=21 дні.

$$З_о = 17500 \cdot 5 / 21 = 3977 \text{ грн.}$$

Проведені розрахунки зведемо до таблиці 4.4.

Таблиця 4.4 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	17500	795,5	5	3977
Інженер-програміст	15000	681,8	40	27273
Всього				31250

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР на тему "Інформаційна технологія оптичного розпізнавання тексту" розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i, \tag{4.2}$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}}, \tag{4.3}$$

де M_M – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо $Мм$=6500 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду (табл. Б.2, додаток Б) [20];

$К_с$ – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об'єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

$Т_р$ – середнє число робочих днів в місяці, приблизно $Т_р$ = 21 дн (табл. 4.5);

$t_{зм}$ – тривалість зміни, год.

$С_1 = 6500,00 \cdot 1,1 \cdot 1,65 / (21 \cdot 8) = 72,4$ грн.

$З_{р1} = 72,4 \cdot 2 = 144,8$ грн.

Проведені розрахунки зведемо до таблиці 4.5.

Таблиця 4.5 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
1. Планування	2	2	72,4	144,8
2. Впровадження	9	5	111,9	1006,8
3. Налагоджувальні	2	2	72,4	144,8
4. Випробувальні	4	1	65,8	263,2
5. Розширення	4	2	72,4	289,5
Всього				1849,1

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (4.4)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 11%.

$З_{дод} = (31250 + 1849,1) \cdot 11 / 100\% = 3640,9$ грн.

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (4.5)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

Зн = (31250+ 1849,1+3640,9) · 22 / 100% = 8082,79 грн.

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень за темою «Інформаційна технологія оптичного розпізнавання тексту».

Витрати на матеріали (*М*), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$М = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} В_j \cdot Ц_{вj}, \quad (4.6)$$

де $Н_j$ – норма витрат матеріалу *j*-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу *j*-го найменування, грн/кг;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ = *1,1 … 1,15*);

$В_j$ – маса відходів *j*-го найменування, кг;

$Ц_{вj}$ – вартість відходів *j*-го найменування, грн/кг.

Проведені розрахунки зведемо до таблиці 4.6.

Таблиця 4.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг, грн	Норма витрат, кг	Вартість витраченого матеріалу, грн
Папір А 4	175	1	175
Ручка	50	1	50
Диск оптичний OPTIMA CD	14	2	28
Flesh-пам'ять GOODRAM 64 C10A	360	1	360
Всього			613
З врахуванням коефіцієнта транспортування			674,3

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування необхідного для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення. В роботі спеціальне устаткування не використовувалося

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для проведення досліджень, також витрати на їх проектування, формування та встановлення.

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \ , \qquad (4.7)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, ($K_i = 1{,}10…1{,}12$);

k – кількість найменувань програмних засобів.

$B_{прг} = 1000 \cdot 3 \cdot 1,11 = 3300$ *грн.*

Отримані результати зведемо до таблиці 4.7:

Таблиця 4.7 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Використання зовнішніх нейромереж і API	3	1000	3300
Всього			3300

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \quad (4.8)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$А_{обл} = (75000 \cdot 1) / (2 \cdot 12) = 3125$ грн.

Проведені розрахунки зведемо до таблиці 4.8.

Таблиця 4.8– Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Macbook Pro 14	75000	2	1	3125,00
Приміщення лабораторії	245000	20	1	1020,83
Всього				4145,83

Витрати на силову електроенергію (B_e) розраховуємо за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{вni}}{\eta_i}, \quad (4.9)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_e$ = 7,5 грн;

$K_{вni}$ – коефіцієнт, що враховує використання потужності, $K_{вni} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$B_e = 0,25 \cdot 260,0 \cdot 7,5 \cdot 0,5 / 0,8 = 310,55$ *грн.*

До статті «Службові відрядження» дослідної роботи на тему "Інформаційна технологія оптичного розпізнавання тексту" належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов’язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з’їзди, конференції, наради, пов’язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100\%}, \quad (4.10)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $Н_{св}$ = 20%.

$В_{св}$ = (31250+ 1849,1) · 20 / 100% = 6619,82 грн.

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (4.11)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%.

$І_в$ = (31250+ 1849,1) · 50 / 100% = 16549,54 грн.

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (4.12)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 100%.

$В_{нзв}$ = (31250+ 1849,1) · 100 / 100% = 33099,08 грн.

Витрати на проведення науково-дослідної роботи на тему "Інформаційна технологія оптичного розпізнавання тексту". розраховуємо як суму всіх попередніх статей витрат за формулою:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + M + К_в + B_{спец} + B_{прг} + A_{обл} + B_е + B_{св} + B_{сп} + I_в + B_{нзв}. \quad (4.13)$$

$В_{заг}$= 31250+ 1849,1+3640,9+8082,79+674,3+3300+4145,83+310,55 +
6619,82 +16549,54+33099,08= 109521,89 грн.

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{B_{заг}}{\eta}, \quad (4.14)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,9.

ЗВ = 109521,89 / 0,9 = 121690,99 грн.

4.4 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження проведені за темою "Інформаційна технологія оптичного розпізнавання тексту" передбачають комерціалізацію протягом 3-х років реалізації на ринку.

В цьому випадку основу майбутнього економічного ефекту будуть формувати:

ΔN – збільшення кількості споживачів яким надається відповідна інформаційна послуга у періоди часу, що аналізуються;

N – кількість споживачів яким надавалась відповідна інформаційна послуга у році до впровадження результатів нової науково-технічної розробки, приймемо 1 особа

$Ц_б$ – вартість послуги у році до впровадження інформаційної системи, приймемо 1200,00 грн;

$\pm\Delta Ц_о$ – зміна вартості послуги від впровадження результатів, приймемо зростання на 450,00 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 3-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою [20]:

$$\Delta П_i = (\pm\Delta Ц_о \cdot N + Ц_о \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (4.15)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2021 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0,8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту). Приймемо ρ =40%;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2023 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1$ =(1·450+1200·700)·0,83·0,4·(1-0,18/100%)=197381,48 грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2$ =(1·450+1200·(700+400))·0,83·0,4·(1-0,18/100%)=310500,1 грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3$ =(1·450+1200·(700+400+200))·0,83·0,4·(1-0,18/100%)=366872,84 грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \quad (4.16)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ=18%;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$ПП = 197381{,}48 /(1+0{,}18)^1 + 310500{,}1 /(1+0{,}18)^2 + 366872{,}84 /(1+0{,}18)^3 =$$
$$= 593407{,}79 \text{ грн.}$$

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \cdot ЗВ, \quad (4.17)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=2;

$ЗВ$ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 121690,99 грн.

$PV = k_{інв} \cdot ЗВ = 2 \cdot 121690{,}99 = 243381{,}98$ грн.

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \quad (4.18)$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 593407,79 грн;

PV – теперішня вартість початкових інвестицій, 243381,98 грн.

$E_{абс} = ПП - PV$ = 593407,79 - 243381,98 = 350025,82 грн.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (4.19)$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, грн;

PV – теперішня вартість початкових інвестицій, грн;

$T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 3 роки.

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1 = (1+350025,82 /243381,98)^{1/3} - 1 = 0,57.$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f, \quad (4.20)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2023 році в Україні d =0,1;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,25.

$\tau_{мін} = 0,1+0,25 = 0,35 < 0,57$ свідчить про те, що внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Інформаційна технологія онтологічного моделювання бази знань з організації бібліотеки» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \tag{4.21}$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок} = 1 / 0,57 = 1,8$ р.

$T_{ок} <$ 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

4.5 Висновки до розділу 4

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Інформаційна технологія оптичного розпізнавання тексту» становить 33 бали, що, свідчить про комерційну важливість проведення даних досліджень оскільки рівень комерційного потенціалу розробки вище середнього.

Також термін окупності становить 1,8 роки, що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою "Інформаційна технологія оптичного розпізнавання тексту".

ВИСНОВКИ

В результаті виконання магістерської кваліфікаційної роботи розроблено інформаційну технологію оптичного розпізнавання тексту.

В роботі була поставлена задача збільшення достовірності сучасного процесу розпізнавання символів. Були розглянуті основні методи та програмні засоби, які виконують розпізнавання символів та визначені їх недоліки. Визначено: основні вимоги до інформаційної технології, вхідні та вихідні дані для програмних засобів та вимоги до програмно-апаратного забезпечення. В результаті аналізу систем аналогів визначено їхні характеристики, переваги та недоліки. Для усунення цих недоліків було запропоновано використовувати метод розпізнавання символів, заснований на використанні штучних нейронних мереж та так званих лінійних сенсорів. В роботі досліджено і розроблено основні етапи, з яких складається процес розпізнавання символів. Було обґрунтовано вибір типу нейронної мережі, розглянуто архітектуру та математичну модель нейронної мережі багатошаровий персептрон, яка навчається за алгоритмов зворотного поширення помилки. Проаналізовано роботу нейронної мережі багатошаровий персептрон в режимі розпізнавання символів. У підсумку було розроблено структуру інформаційної технології розпізнавання символів на основі лінійних сенсорів та багатошарового персептрона, розроблено алгоритм роботи та UML-діаграму класів програмного забезпечення розпізнавання символів.

В ході програмної реалізації інформаційної технології розпізнавання тексту розглянуто та обґрунтовано переваги та недоліки об’єктно-орієнтованої мови програмування високого рівня C# та середовища Microsoft Visual Studio. В результаті було спроектовано власне програмне забезпечення розпізнавання символів. Програму написано об’єктно-орієнтованою мовою програмування високого рівня C# з використанням Windows Forms під операційну систему Windows. Програма має віконний інтерфейс. Користувач може обирати тип, вид написання, розмір та колір шрифта, який береться як еталонний. Також може малювати зображення символу, який підлягає розпізнаванню, розпізнавати намальоване зображення символу за допомогою нейронної мережі. В результаті даної роботи отримано інтелектуальний

програмний продукт розпізнавання символів на основі нейронної мережі багатошаровий персептрон, який порівняно з аналогом має кращу на 4,9% достовірність розпізнавання символів (96,7%), ніж аналогічна програма (91,8%).

В розділі економічної частини проведено оцінювання комерційного потенціалу розробки інформаційної технології оптичного розпізнавання тексту, спрогнозовано витрати на виконання наукової роботи та впровадження результатів, які склали 121690,99 грн, розраховано період окупності – 1,8 року.

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Інформаційна технологія оптичного розпізнавання тексту» становить 33 бали, що, свідчить про комерційну важливість проведення даних досліджень оскільки рівень комерційного потенціалу розробки вище середнього.

Також термін окупності становить 1,8 роки, що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Таким чином, мета роботи досягнута – достовірність розпізнавання символів підвищена.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. С.В. Барабан, О.О. Горбатюк. Структура інформаційної технології розпізнавання тексту. Матеріали Всеукраїнської науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2023)» – [Електронний ресурс]. – https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2023/paper/view/19453

2. Explanation of basic handwriting recognition principles and history. [Електронний ресурс] – Режим доступу: http://technology.17things.com /what-is-handwriting-recognition.html

3. Жосан А. А. Алгоритми побудови комп'ютерної системи розпізнавання образів з використанням паралельних нейромережних обчислень [Електронний ресурс] / А. А. Жосан. – Режим доступу: http://www.rusnauka.com/2_KAND_2008/Informatica/26121.doc.htm

4. Руденко О.В. Штучні нейронні мережі: Навчальний посібник / О.В.Руденко, Є.В.Бодянський. - Харків : ТОВ «Компанія СМІТ», 2006. — 404 с. - ISBN 966-8630-73-Х.

5. Куссуль Н. М. Інтелектуальні обчислення: навчальний посібник (із грифом МОН України) / Н. М. Куссуль, А. Ю. Шелестов, А. Н. Лавренюк. - К. : «Наукова думка», 2006. — 186 с. ISBN 966-00-0592-Х

6. Літанова Є.П. Мова UML. Її зв'язок з сучасними підходами до розробки програмних систем. [Електронний ресурс]/ Є. П. Літанова. – Режим доступу: http://webkonspect.com/?room=group&id=7&labelid=169

7. Програма розпізнавання тексту CuneiForm // [Електронний ресурс]. - Режим доступу: http://www.folksoft.net/soft/apps/cuneiform.html/

8. How to change the color of progressbar in C# .NET 3.5? Режим доступу: https://stackoverflow.com/questions/778678/how-to-change-the-color-of-progressbar-in-c-sharp-net-3-5 (дата звернення черв. 2022).

9. Наконечний С.І., Савіна С.С. Математичне програмування: Навч.посіб. – К.: КНЕУ, 2003. – 452 с.

10. C++ 17 STL. Стандартна бібліотека шаблонів. Яцек Галовіц. 2018. ISBN: 978-5-4461-0680-6

11. Парадигми програмування. Елементи програм C++ [Electronic resource]. – Mode of access : WWW/URL : http://om.univ.kiev.ua/users_upload/15/upload/file/prog_lecture_01.pdf. – Title from the screen.

12. Кравець П. Об'єктно-орієнтоване програмування : навч. посібник / П.О. Кравець. – Львів: Видавництво Львівської політехніки, 2012. – 624 с.

13. C Tutorial [Electronic resource]. – Mode of access : WWW/URL : https://www.tutorialspoint.com/cprogramming/index.htm. – Title from the screen.

14.Josuttis, Nicolai M. The C++ standard library : a tutorial and reference / Nicolai M. Josuttis.—2nd ed. c. 78-80.

15.Metanit Об'єктно-орієнтоване програмування [Електронний ресурс] / Metanit. - Режим доступу: https://metanit.com/sharp/tutorial/3.1.php

16.ITVDN Об’єктно-орієнтоване програмування C# в деталях / ITVDN. – Режим доступу: https://itvdn.com/ua/blog/article/oop-csharp-basic

17.Глинський Я. М., Анохін В. Є., Ряжська В. А. C++ і C++ Builder. Львів : СПД Глинський, 2011.

18.Войтенко В.В. Морозов А.В. Теорія та практика (мова С++). — Житомир, 2002.

19. Методичні вказівки до виконання студентами-магістрантами економічної частини магістерських кваліфікаційних робіт / Уклад. В. О. Козловський – Вінниця: ВНТУ, 2012. – 22 с.

20. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.

ДОДАТКИ

Додаток А (обов'язковий)

Протокол перевірки кваліфікаційної роботи на наявність текстових запозичень

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Інформаційна технологія оптичного розпізнавання тексту

Тип роботи: магістерська кваліфікаційна робота
(БДР, МКР)

Підрозділ кафедра комп'ютерних наук, ФІТА
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 88,2% Схожість 11,8%

Аналіз звіту подібності (відмітити потрібне):

✓ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи Горбатюк О.О.

Керівник роботи Барабан С.В.

Опис прийнятого рішення

Магістерську кваліфікаційну роботу допущено до захисту

Особа, відповідальна за перевірку Озеранський В.С.

Додаток Б (обов'язковий)

Лістинг програми

```
// AForge Neural Net Library
//
// Copyright © 2015
//

namespace AForge.NeuralNet.Learning
{
using System;

/// <summary>
/// Back Propagation learning
/// </summary>
public class BackPropagationLearning : ISupervisedLearning
{
        private Network          net;    // network to learn
        private float            learningRate = 0.1f;
        private float            momentum = 0.0f;
        private float            learningLimit = 0.1f;

        private bool             converged = false;

        private float[][]        errors;
        private float[][][]      deltas;
        private float[][]        thresholdDeltas;

        // Learning Rate property
        public float LearningRate
        {
                get { return learningRate; }
                set { learningRate = value; }
        }
        // Momentum property
        public float Momentum
        {
                get { return momentum; }
                set { momentum = value; }
        }
        // Learning Limit property
        public float LearningLimit
        {
                get { return learningLimit; }
                set { learningLimit = value; }
        }
        // Is converged property
        public bool IsConverged
        {
                get { return converged; }
        }

        // Constructor
        public BackPropagationLearning(Network net)
        {
                this.net = net;

                // create error and deltas arrays
                errors = new float[net.LayersCount][];
                deltas = new float[net.LayersCount][][];
                thresholdDeltas = new float[net.LayersCount][];

                for (int i = 0; i < net.LayersCount; i++)
                {
                        Layer layer = net[i];

                        errors[i] = new float[layer.NeuronsCount];
                        deltas[i] = new float[layer.NeuronsCount][];
                        thresholdDeltas[i] = new float[layer.NeuronsCount];

                        for (int j = 0; j < layer.NeuronsCount; j++)
                        {
                                deltas[i][j] = new float[layer.InputsCount];
```

```
                }
        }
}

// Perform learning epoch and return errors sum
public float LearnEpoch(float[][] input, float[][] output)
{
        int             i, n = input.Length;
        float   error = 0.0f;

        // for all training patterns
        for (i = 0; i < n; i++)
        {
                error += Learn(input[i], output[i]);
        }
        // determine if we converged
        converged = (error < learningLimit);

        // return error
        return error;
}

// Perform one learning iteration and return network error
public float Learn(float[] input, float[] output)
{
        // compute the network
        float[] nout = net.Compute(input);

        // calculate network error
        float error = CalculateError(output);

        // calculate weights updates
        CalculateUpdates(input);

        // update the network
        UpdateNetwork();

        // return error level
        return error;
}

// Calculate network errors
private float CalculateError(float[] desiredOutput)
{
        Layer           layer, layerNext;
        float[]         err, errNext;
        float           error = 0, e;
        float           output, sum;
        int                     i, j, k, n, m, layersCount = net.LayersCount;

        // assume, that all neurons of the network have
        // the same activation function
        IActivationFunction     function = net[0][0].ActivationFunction;

        // calculate error for the last layer
        layer = net[layersCount - 1];
        err = errors[layersCount - 1];

        for (i = 0, n = layer.NeuronsCount; i < n; i++)
        {
                output = layer[i].Output;
                // error of the neuron
                e = desiredOutput[i] - output;
                // error multiplied with first derivative
                err[i] = e * function.OutputPrime2(output);
                // squre the error and sum it
                error += (e * e);
        }

        // calculate error for other layers
        for (j = layersCount - 2; j >= 0; j--)
        {
                layer           = net[j];
                layerNext       = net[j + 1];
                err                     = errors[j];
                errNext         = errors[j + 1];
```

```
                // for all neurons of the layer
                for (i = 0, n = layer.NeuronsCount; i < n; i++)
                {
                        sum = 0.0f;
                        // for all neurons of the next layer
                        for (k = 0, m = layerNext.NeuronsCount; k < m; k++)
                        {
                                sum += errNext[k] * layerNext[k][i];
                        }
                        err[i] = sum * function.OutputPrime2(layer[i].Output);
                }
        }

        // return squared error of the last layer
        return error;
}

// Calculate synapse (neurons weights) updates
private void CalculateUpdates(float[] input)
{
        Neuron          neuron;
        Layer           layer, layerPrev;
        float[][]       lDeltas;
        float[]         err, del, tdel;
        float           e;
        int                     i, j, k, n, m, l;

        // 1 - for the first layer
        layer   = net[0];
        lDeltas = deltas[0];
        err             = errors[0];
        tdel    = thresholdDeltas[0];

        // for each neuron of the layer
        for (i = 0, n = layer.NeuronsCount; i < n; i++)
        {
                neuron  = layer[i];
                del             = lDeltas[i];
                e               = err[i];

                // for each synapse of the neuron
                for (j = 0, m = neuron.InputsCount; j < m; j++)
                {
                        // calculate weight update
                        del[j] = learningRate * (
                                        momentum * del[j] +
                                        (1.0f - momentum) * e * input[j]
                                );
                }

                // calculate treshold update
                tdel[i] = learningRate * (
                        momentum * tdel[i] +
                        (1.0f - momentum) * e
                        );
        }

        // 2 - for all other layers
        for (k = 1, l = net.LayersCount; k < l; k++)
        {
                layerPrev = net[k - 1];
                layer   = net[k];
                lDeltas = deltas[k];
                err             = errors[k];
                tdel    = thresholdDeltas[k];

                // for each neuron of the layer
                for (i = 0, n = layer.NeuronsCount; i < n; i++)
                {
                        neuron  = layer[i];
                        del             = lDeltas[i];
                        e               = err[i];

                        // for each synapse of the neuron
                        for (j = 0, m = neuron.InputsCount; j < m; j++)
                        {
                                // calculate weight update
```

```
                                        del[j] = learningRate * (
                                                momentum * del[j] +
                                                (1.0f - momentum) * e * layerPrev[j].Output
                                                );
                                }

                                // calculate treshold update
                                tdel[i] = learningRate * (
                                        momentum * tdel[i] +
                                        (1.0f - momentum) * e
                                        );
                        }
                }
        }

        // Update weights of network
        private void UpdateNetwork()
        {
                Neuron          neuron;
                Layer           layer;
                float[][]       lDeltas;
                float[]         del, tdel;
                int                     i, j, k, n, m, s;

                // for each layer of the network
                for (i = 0, n = net.LayersCount; i < n; i++)
                {
                        layer = net[i];
                        lDeltas = deltas[i];
                        tdel = thresholdDeltas[i];

                        // for each neuron of the layer
                        for (j = 0, m = layer.NeuronsCount; j < m; j++)
                        {
                                neuron = layer[j];
                                del = lDeltas[j];

                                // for each weight of the neuron
                                for (k = 0, s = neuron.InputsCount; k < s; k++)
                                {
                                        // update weight
                                        neuron[k] += del[k];
                                }
                                // update treshold
                                neuron.Threshold -= tdel[j];
                        }
                }
        }
}
}

        // On "Create Network" button
        private void createNetButton_Click(object sender, System.EventArgs e)
        {
                CreateNetwork();

                // enable train and recognize buttons
                traintNetworkButton.Enabled = true;
                recognizeButton.Enabled = true;

                //
                errorBox.Text = string.Empty;
                misclassifiedBox.Text = string.Empty;
                outputBox.Text = string.Empty;
        }

        // Create Network
        private void CreateNetwork()
        {
                if (data == null)
                        return;

                int             objectsCount = data.GetLength(0);
                int             featuresCount = data.GetLength(1);
                float   alfa;

                // get alfa value
```

```
                        try
                        {
                                alfa = Math.Max(0.1f, Math.Min(10.0f, float.Parse(alfaBox.Text)));
                        }
                        catch (Exception)
                        {
                                alfa = 1.0f;
                        }
                        alfaBox.Text = alfa.ToString();

                        // creare network
                        if (layersCombo.SelectedIndex == 0)
                        {
                                neuralNet = new Network(new BipolarSigmoidFunction(alfa), featuresCount,
objectsCount);
                        }
                        else
                        {
                                neuralNet = new Network(new BipolarSigmoidFunction(alfa), featuresCount,
objectsCount, objectsCount);
                        }

                        // randomize network`s weights
                        neuralNet.Randomize();
                }


                // On "Traing" button click
                private void traintNetworkButton_Click(object sender, System.EventArgs e)
                {
                        outputBox.Text = string.Empty;

                        // get parameters
                        try
                        {
                                learningEpoch    = Math.Max(0, int.Parse(learningEpochBox.Text));
                                learningRate1    = Math.Max(0.0001f, Math.Min(10.0f, float.Parse(rate1Box.Text)));
                                errorLimit1              = Math.Max(0.0001f, Math.Min(1000.0f,
float.Parse(limit1Box.Text)));
                                learningRate2    = Math.Max(0.0001f, Math.Min(10.0f, float.Parse(rate2Box.Text)));
                                errorLimit2              = Math.Max(0.0001f, Math.Min(1000.0f,
float.Parse(limit2Box.Text)));
                        }
                        catch (Exception)
                        {
                                learningEpoch    = 0;
                                learningRate1    = 1.0f;
                                errorLimit1              = 1.0f;
                                learningRate2    = 0.2f;
                                errorLimit2              = 0.1f;
                        }
                        learningEpochBox.Text = learningEpoch.ToString();
                        rate1Box.Text    = learningRate1.ToString();
                        limit1Box.Text   = errorLimit1.ToString();
                        rate2Box.Text    = learningRate2.ToString();
                        limit2Box.Text   = errorLimit2.ToString();

                        //
                        workType = 1;
                        progressBar.Hide();

                        // start work
                        StartWork(true);

                        // set status message
                        statusBox.Text = "Training network ...";

                        // create and start new thread
                        workerThread = new Thread(new ThreadStart(TrainNetwork));
                        // start thread
                        workerThread.Start();
                }

                // Traing neural network to recognize our training set
                private void TrainNetwork()
                {
                        if (data == null)
```

```
            return;

        int             objectsCount = data.GetLength(0);
        int             featuresCount = data.GetLength(1);
        int             variantsCount = data[0, 0].Length;
        int             i, j, k, n;

        // generate possible outputs
        float[][]       possibleOutputs = new float[objectsCount][];

        for (i = 0; i < objectsCount; i++)
        {
            possibleOutputs[i] = new float[objectsCount];
            for (j = 0; j < objectsCount; j++)
            {
                possibleOutputs[i][j] = (i == j) ? 0.5f : -0.5f;
            }
        }

        // generate network training data
        float[][]       input = new float [objectsCount * variantsCount][];
        float[][]       output = new float [objectsCount * variantsCount][];
        float[]         ins;

        // for all varaints
        for (j = 0, n = 0; j < variantsCount; j++)
        {
            // for all objects
            for (i = 0; i < objectsCount; i++, n++)
            {
                // prepare input
                input[n] = ins = new float[featuresCount];

                // for each receptor
                for (k = 0; k < featuresCount; k++)
                {
                    ins[k] = (float) data[i, k][j] - 0.5f;
                }

                // set output
                output[n] = possibleOutputs[i];
            }
        }

        System.Diagnostics.Debug.WriteLine("--- learning started");

        // create network teacher
        BackPropagationLearning  teacher = new BackPropagationLearning(neuralNet);

        // First pass
        teacher.LearningLimit    = errorLimit1;
        teacher.LearningRate     = learningRate1;

        i = 0;
        // learn
        do
        {
            error = teacher.LearnEpoch(input, output);
            i++;

            // report status
            if ((i % 100) == 0)
            {
                ReportProgress(0, string.Format("Learning, 1st pass ... (iterations: {0},
error: {1})",
                    i, error));
            }

            // need to stop ?
            if (stopEvent.WaitOne(0, true))
                break;
        }
        while (((learningEpoch == 0) && (!teacher.IsConverged)) ||
            ((learningEpoch != 0) && (i < learningEpoch)));

        System.Diagnostics.Debug.WriteLine("first pass: " + i + ", error = " + error);
```

```
                        // skip second pass, if learning epoch number was specified
                        if (learningEpoch == 0)
                        {
                                // Second pass
                                teacher = new BackPropagationLearning(neuralNet);

                                teacher.LearningLimit    = errorLimit2;
                                teacher.LearningRate     = learningRate2;

                                // learn
                                do
                                {
                                        error = teacher.LearnEpoch(input, output);
                                        i++;

                                        // report status
                                        if ((i % 100) == 0)
                                        {
                                                ReportProgress(0, string.Format("Learning, 2nd pass ...
(iterations: {0}, error: {1})",
                                                        i, error));
                                        }

                                        // need to stop ?
                                        if (stopEvent.WaitOne(0, true))
                                                break;
                                }
                                while (!teacher.IsConverged);

                                System.Diagnostics.Debug.WriteLine("second pass: " + i + ", error = " + error);
                        }

                        // get the misclassified value
                        misclassified = 0;
                        // for all training patterns
                        for (i = 0, n = input.Length; i < n; i++)
                        {
                                float[] realOutput = neuralNet.Compute(input[i]);
                                float[] desiredOutput = output[i];
                                int             maxIndex1 = 0;
                                int             maxIndex2 = 0;
                                float   max1 = realOutput[0];
                                float   max2 = desiredOutput[0];

                                for (j = 1, k = realOutput.Length; j < k; j++)
                                {
                                        if (realOutput[j] > max1)
                                        {
                                                max1 = realOutput[j];
                                                maxIndex1 = j;
                                        }
                                        if (desiredOutput[j] > max2)
                                        {
                                                max2 = desiredOutput[j];
                                                maxIndex2 = j;
                                        }
                                }

                                if (maxIndex1 != maxIndex2)
                                        misclassified++;
                        }
                }

                // On "Recognize" button click
                private void recognizeButton_Click(object sender, System.EventArgs e)
                {
                        int             i, n, maxIndex = 0;

                        // get current receptors state
                        int[]   state = receptors.GetReceptorsState(paintBoard.GetImage());

                        // for network input
                        float[] input = new float[state.Length];

                        for (i = 0; i < state.Length; i++)
                                input[i] = (float) state[i] - 0.5f;
```

```
                // compute network and get it's ouput
                float[] output = neuralNet.Compute(input);

                // find the maximum from output
                float max = output[0];
                for (i = 1, n = output.Length; i < n; i++)
                {
                        if (output[i] > max)
                        {
                                max = output[i];
                                maxIndex = i;
                        }
                }

                //
                outputBox.Text = string.Format("{0}", (char)((int) 'A' + maxIndex));
        }

        // On "Stop" button click - stop the work
        private void stopButton_Click(object sender, System.EventArgs e)
        {
                if (stopEvent != null)
                        stopEvent.Set();
        }

        // On "Scale" checkbox changed
        private void scaleCheck_CheckedChanged(object sender, System.EventArgs e)
        {
                paintBoard.ScaleImage = scaleCheck.Checked;
        }
}

private void RemoveLearningDuplicates()
        {
                if (data == null)
                        return;

                int             objectsCount = data.GetLength(0);
                int             featuresCount = data.GetLength(1);
                int             variantsCount = data[0, 0].Length;

                int             i, j, k, s;
                int[]   item;

                // calculate checksum of each object for each receptor
                int[,]  checkSum = new int[objectsCount, featuresCount];

                // for each object
                for (i = 0; i < objectsCount; i++)
                {
                        // for each receptor
                        for (j = 0; j < featuresCount; j++)
                        {
                                item = data[i, j];
                                s = 0;

                                // for each variant
                                for (k = 0; k < variantsCount; k++)
                                {
                                        s |= item[k] << k;
                                }

                                checkSum[i, j] = s;
                        }
                }

                // find which receptors should be removed
                bool[]  remove = new bool[featuresCount];

                // walk through all receptors ...
                for (i = 0; i < featuresCount - 1; i++)
                {
                        // skip receptors alredy marked as deleted
                        if (remove[i] == true)
                                continue;

                        // ... and compare each receptor with others
```

```
                for (j = i + 1; j < featuresCount; j++)
                {
                        // remove by default
                        remove[j] = true;

                        // compare cheksums of all objects
                        for (k = 0; k < objectsCount; k++)
                        {
                                if (checkSum[k, i] != checkSum[k, j])
                                {
                                        // ups, they are different, do not delete it
                                        remove[j] = false;
                                        break;
                                }
                        }
                }
        }

        // count receptors to save
        int receptorsToSave = 0;
        for (i = 0; i < featuresCount; i++)
                receptorsToSave += (remove[i]) ? 0 : 1;

        // filter data removing receptors with usability below acceptable
        int[,][] newData = new int [objectsCount, receptorsToSave][];
        Receptors newReceptors = new Receptors();

        k = 0;
        // for all receptors
        for (j = 0; j < featuresCount; j++)
        {
                if (remove[j])
                        continue;

                // for all objects
                for (i = 0; i < objectsCount; i++)
                {
                        newData[i, k] = data[i, j];
                }
                newReceptors.Add(receptors[j]);
                k++;
        }

        // set new data
        data = newData;
        receptors = newReceptors;
}

// Filter learning data
private void FilterLearningData()
{
        if (data == null)
                return;

        // data filtering is performed by removing bad receptors

        int             objectsCount = data.GetLength(0);
        int             featuresCount = data.GetLength(1);
        int             variantsCount = data[0, 0].Length;
        int             i, j, k, v;
        int[]   item;

        // maybe we already filtered ?
        // so check that new receptors count is not greater than we have
        if (receptorsCount >= featuresCount)
                return;

        int[]   outerCounters = new int[2];
        int[]   innerCounters = new int[2];
        double  ie, oe;

        double[] usabilities = new double[featuresCount];

        // for all receptors
        for (j = 0; j < featuresCount; j++)
        {
                // clear outer counters
```

```
        Array.Clear(outerCounters, 0, 2);

        ie = 0;
        // for all objects
        for (i = 0; i < objectsCount; i++)
        {
                // clear inner counters
                Array.Clear(innerCounters, 0, 2);
                // get variants item
                item = data[i, j];

                // for all variants
                for (k = 0; k < variantsCount; k++)
                {
                        v = item[k];

                        innerCounters[v]++;
                        outerCounters[v]++;
                }

                // callculate inner entropy of receptor for current object
                ie += Statistics.Entropy(innerCounters, variantsCount);
        }

        // average inner entropy
        ie /= objectsCount;
        // outer entropy
        oe = Statistics.Entropy(outerCounters, objectsCount * variantsCount);
        // receptors usability
        usabilities[j] = (1.0 - ie) * oe;
}

// create usabilities copy and sort it
double[] temp = (double[]) usabilities.Clone();
Array.Sort(temp);
// get acceptable usability for receptor
double accaptableUsability = temp[featuresCount - receptorsCount];

// filter data removing receptors with usability below acceptable
int[,][] newData = new int [objectsCount, receptorsCount][];
Receptors newReceptors = new Receptors();

k = 0;
// for all receptors
for (j = 0; j < featuresCount; j++)
{
        if (usabilities[j] < accaptableUsability)
                continue;

        // for all objects
        for (i = 0; i < objectsCount; i++)
        {
                newData[i, k] = data[i, j];
        }
        newReceptors.Add(receptors[j]);

        if (++k == receptorsCount)
                break;
}

// set new data
data = newData;
receptors = newReceptors;}
```

Додаток В (обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ОПТИЧНОГО РОЗПІЗНАВАННЯ ТЕКСТУ

Виконав: студент 2 курсу, групи 2КН-22м
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Горбатюк О.О.
(прізвище та ініціали)
Керівник: к.т.н., доцент каф. КН
Барабан С.В.
(прізвище та ініціали)
« 07 » 12 2023 р.

Вінниця ВНТУ – 2023 рік

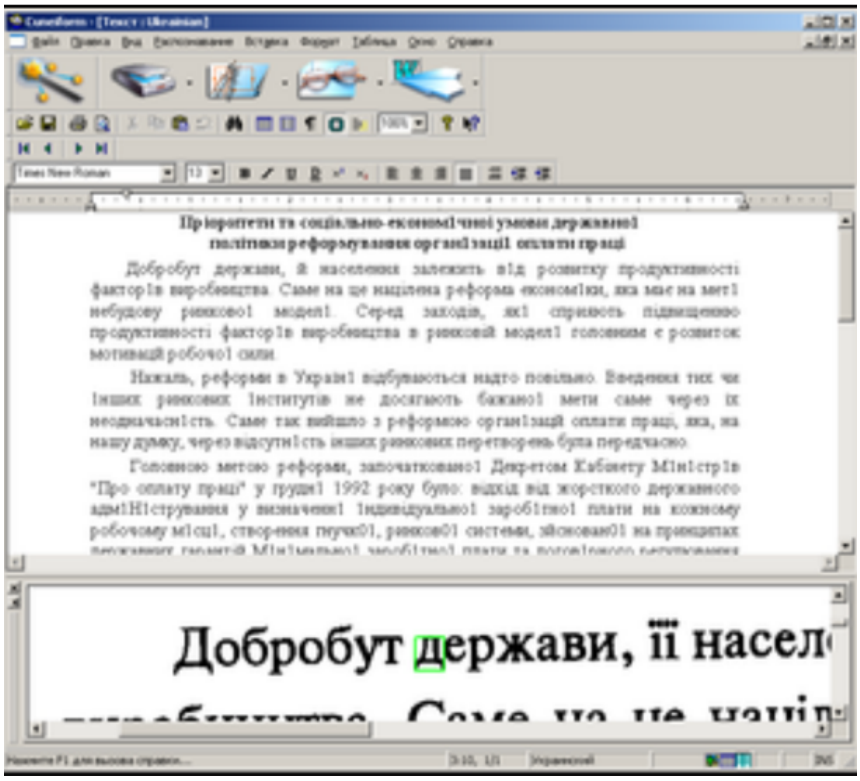

Рисунок В.1 – Програми аналоги розпізнавання тексту

	0	1	2
0	11101	00000	00000
1	11111	00000	01011
2	11101	00000	00000
3	11111	00010	11111
4	10101	00000	00000

Рисунок В.2 – Метод отримання ознак текстового символу на основі лінійних сенсорів

Рисунок В.3 – Структура інформаційної технології оптичного розпізнавання тексту

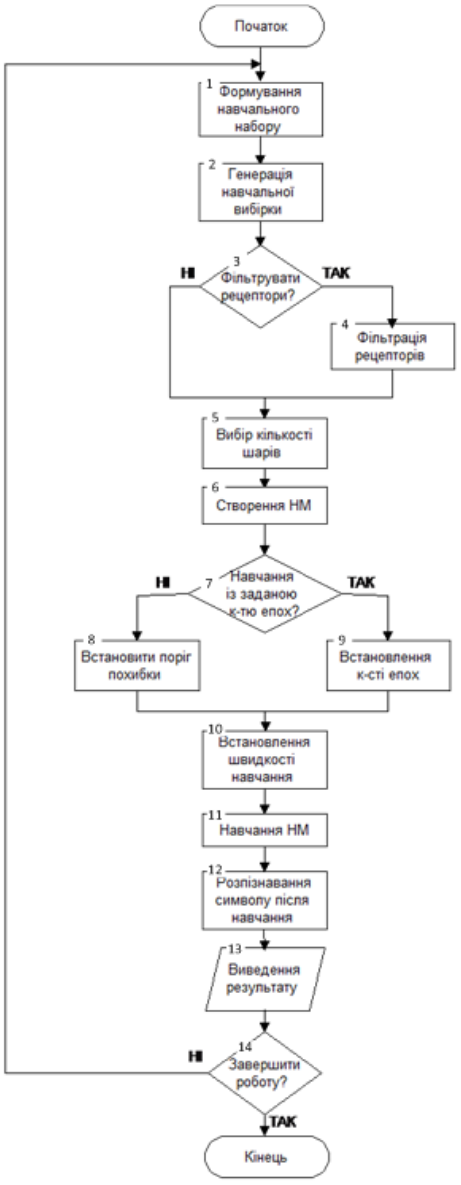

Рисунок В.4 – Алгоритм роботи інформаційної технології оптичного розпізнавання тексту

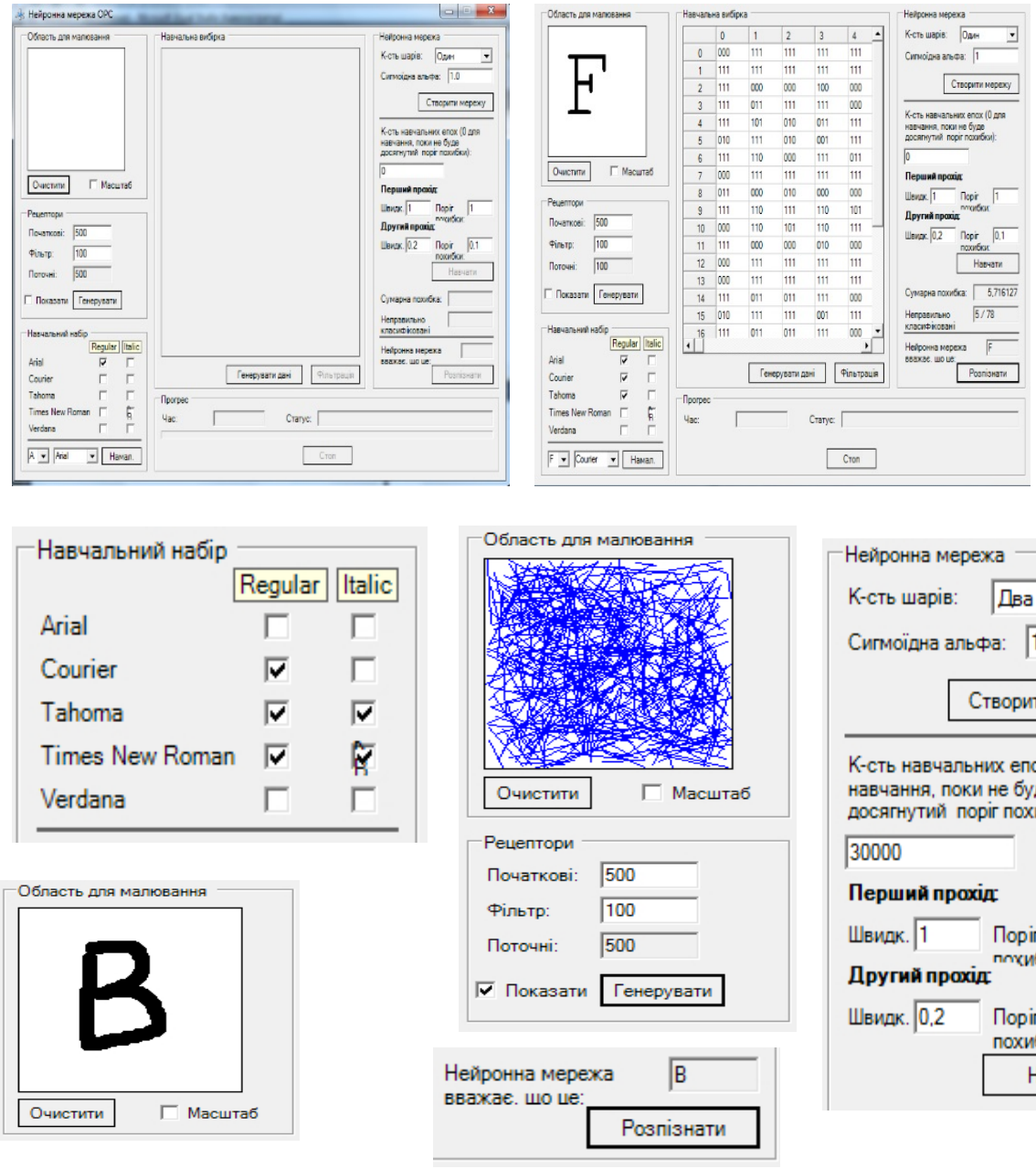

Рисунок В.5 – Програмна реалізація інформаційної технологій оптичного розпізнавання тексту

Додаток Г (довідниковий)

Інструкція користувача

Створена програма розпізнавання тексту не вимагає інсталяції. Для того, щоб розпочати роботу з програмою розпізнавання тексту потрібно лише скопіювати папку з програмою та запустити файл NewOCR.exe. Головне вікно програми зображено на рис. Г.1.

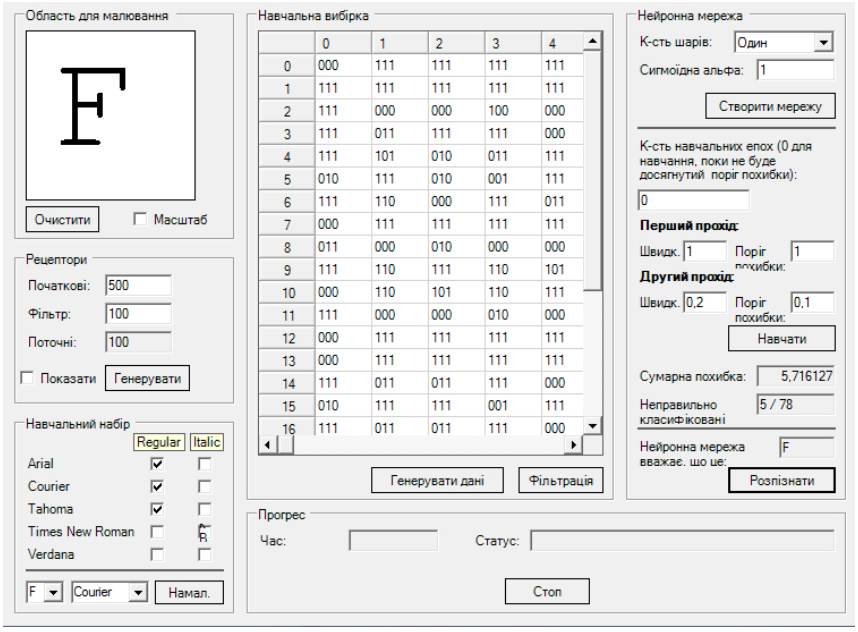

Рисунок Г.1 – Головне вікно програми розпізнавання тексту

Далі потрібно в блоці «Навчальний набір» відмітити галочками усі типи шрифтів, які наша нейронна мережа буде навчатись розпізнавати. Серед доступних в програмі шрифтів є такі: Arial, Courier, Tahoma, Times New Roman, Verdana. Для кожного з шрифтів можна вибрати його тип: стандартний або курсивний. Вибір шрифтів зображений на рис. Г.2.

Потім необхідно обрати кількість сенсорів, що будуть відповідати за якість навчання та розпізнавання символів. За замовчуванням, кількість початкових сенсорів дорівнює 500. Також потрібно обрати кількість сенсорів, які залишаться після їх фільтрації. За замовчуванням, їх кількість рівна 100.

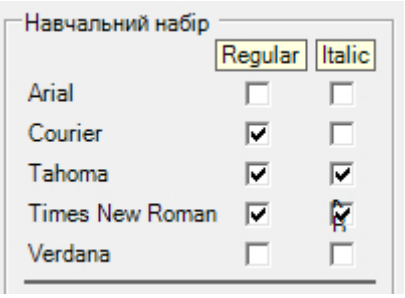

Рисунок Г.2 – Блок програми, що відповідає за навчальний набір шрифтів

Потім потрібно натиснути кнопку «Генерувати» і програма згенерує випадковим чином набір сенсорів. Побачити їх в області для малювання можна, відмітивши галочку «Показати». Згенеровані програмою сенсори зображені на рис. Г.3.

Рисунок Г.3 – Згенеровані програмою розпізнавання символів сенсори

Далі потрібно натиснути кнопку «Генерувати дані», щоб програма сформувала набір даних для навчання нейронної мережі. Кількість навчальних даних можна зменшити відібравши лише якісні сенсори. Для цього потрібно натиснути кнопку «Фільтрація».

Далі в блоці «Нейронна мережа» необхідно вибрати кількість шарів нейронної мережі (в програмі доступні 1 або 2 шари) та значення сигмоїдної альфи функції активації, і натиснути кнопку «Створити мережу».

Потім потрібно вказати кількість епох, протягом яких буде проводитись навчання або залишити 0 для навчання, поки не буде досягнутий поріг похибки. Нижче вказати швидкість першого та другого проходу навчання, а також значення похибки, при якій навчання буде зупинятись. Потім натиснути кнопку «Навчати» та очікувати поки нейронна мережа не закінчить навчання. Один з прикладів налаштування параметрів для навчання нейронної мережі зображено на рис. Г.4.

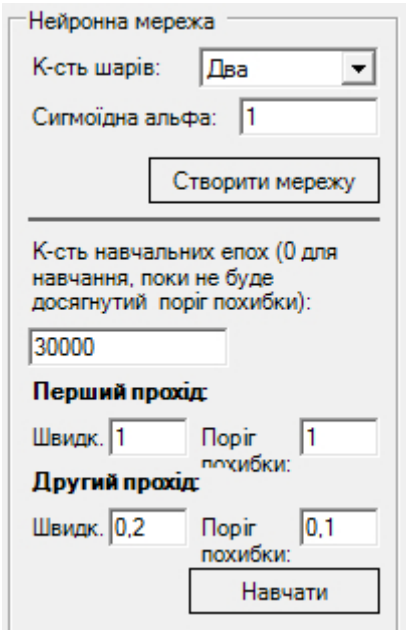

Рисунок Г.4 – Налаштування параметрів навчання нейронної мережі

Процес навчання зображено на рис. Г.5. Навчання можна зупинити в будь-який момент, натиснувши кнопку «Стоп».

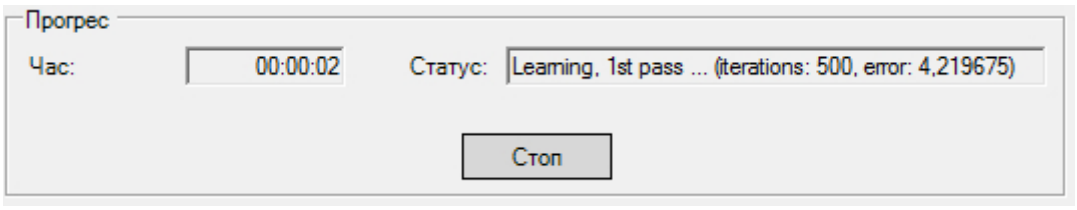

Рисунок А.5 – Процес навчання нейронної мережі

В результаті отримуємо сумарну похибку та кількість неправильно класифікованих символів.

Після навчання потрібно обрати символ для розпізнавання. Це можна зробити двома шляхами: намалювати вручну в області для малювання або вибрати шрифт та символ і програма намалює автоматично. І натиснути кнопку «Розпізнати», після чого мережа виводить результат. Приклад вхідних ти вихідних даних розпізнавання символу зображено на рис. Г.6 і Г.7 відповідно.

Рисунок Г.6 – Намальований вручну текстовий символ для розпізнавання

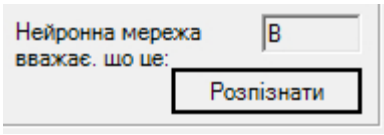

Рисунок Г.7 – Результат розпізнавання програмою намальованого текстового символу