

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістр

(ступінь вищої освіти)

на тему РОЗРОБКА ТА ДОСЛІДЖЕННЯ АЛГОРИТМІВ
МАШИННОГО НАВЧАННЯ ДЛЯ РОЗПІЗНАВАННЯ
РУКОПИСНОГО ТЕКСТУ
DEVELOPMENT AND RESEARCH OF MACHINE
LEARNING ALGORITHMS FOR HANDWRITING
RECOGNITION

Виконав(ла): студент(ка) 2 курсу, групи КНТ-124м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

НЕСТЕРОВ Г.Д.

(ПРИЗВИЩЕ та ініціали)

Керівник ГОФМАН Є.О.

(ПРИЗВИЩЕ та ініціали)

Рецензент ПОЛЯКОВ М.О.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2025 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

НЕСТЕРОВА Геннадія Денисовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проекту (роботи) Розробка та дослідження алгоритмів машинного навчання для розпізнавання рукописного тексту. Development and Research of Machine Learning Algorithms for Handwriting Recognition

керівник проекту (роботи) к.т.н., доцент ГОФМАН Євгеній Олександрович,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 30 ” вересня 2025 року № 447

2. Строк подання студентом проекту (роботи) 08 грудня 2025 року

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Обґрунтування вибору програмних засобів розробки. 3. Основні рішення щодо реалізації компонентів системи. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ГОФМАН Є.О., доцент		
Нормоконтроль	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання “30” вересня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2-3 тижні	Розділ 1
3	Розробка та удосконалення методів, моделей й алгоритмів вирішення задачі.	4-5 тижні	Розділ 2
4	Розробка архітектури програми.	6 тиждень	
5	Розробка програми.	7-8 тижні	Розділ 3
6	Тестування та експериментальне дослідження програмного забезпечення.	9 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	10-11 тижні	Додатки
8	Нормоконтроль та рецензування.	12 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка)

_____ Геннадій НЕСТЕРОВ
(підпис) (Імя ПРИЗВИЩЕ)

Керівник проєкту (роботи)

_____ Євгеній ГОФМАН
(підпис) (Імя ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
88 с., 7 табл., 34 рис., 2 дод., 25 джерел.

РОЗПІЗНАВАННЯ РУКОПИСНОГО ТЕКСТУ, АНАЛІЗ
ЗОБРАЖЕНЬ, МАШИННЕ НАВЧАННЯ, ШТУЧНИЙ ІНТЕЛЕКТ,
ЗГОРТКОВА НЕЙРОННА МЕРЕЖА.

Об'єкт дослідження – процес розробки програмного забезпечення для розпізнавання рукописного тексту.

Предмет дослідження – програмні засоби для реалізації алгоритмів машинного навчання для розпізнавання рукописного тексту.

Мета роботи – створення програмного забезпечення для підвищення точності розпізнавання рукописного тексту.

Матеріали, методи та технічні засоби: html, css, мови програмування Python і JavaScript, TensorFlow, персональний комп'ютер під управлінням операційної системи Microsoft Windows 11.

Результати. Розроблено програмне забезпечення для розпізнавання рукописного тексту.

Актуальність та проблематика роботи полягає в зростаючому попиті на машинне навчання на ринку товарів ІТ галузі і, водночас, цифролізації документообігу і повсякденної передачі текстової інформації.

Наукова новизна одержаних результатів:

- удосконалено методику попередньої обробки рукописних зображень шляхом комбінування адаптивної бінаризації, нормалізації та контурного фільтрування, що забезпечує підвищення якості вхідних даних перед подачею в модель глибокого навчання;

- запропоновано модифіковану архітектуру рекурентної нейронної мережі, оптимізовану для розпізнавання дрібних рукописних символів

шляхом поєднання згорткових блоків із шарами підвищеної чутливості до локальних особливостей зображення та використання довгої короткострокової пам'яті для збереження проміжних результатів розпізнавання;

- удосконалено підхід до навчання моделі за рахунок використання розширеного датасету для навчання, що підвищує стійкість мережі до різних типів рукописів;

- розроблено програмний застосунок із використанням власної моделі машинного навчання, що забезпечує інтеграцію процесів попередньої обробки, розпізнавання та візуалізації результатів.

Висновки. Виконано проєктування та реалізація програмного забезпечення для розпізнавання рукописного тексту, здійснено тестування розробленого програмного забезпечення.

Галузь використання – програмний застосунок може бути застосований в багатьох галузях, зокрема, в організації навчального процесу закладів освіти, переведенні архівних документів державних підприємств в цифровий вигляд тощо.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 88 pages, 7 tables, 34 figures, 2 appendixes, 25 sources.

HANDWRITING RECOGNITION, IMAGE PROCESSING, MACHINE LEARNING, ARTIFICIAL INTELLIGENCE, CONVOLUTIONAL NEURAL NETWORK.

Object of study is the process of developing software for handwriting recognition.

Subject of study are software tools for implementing machine learning algorithms for handwriting recognition.

The purpose of the work is to create a software to improve the accuracy of handwritten text recognition.

Materials, methods, and tools: html, css, Python and JavaScript programming languages, TensorFlow, personal computer running the Microsoft Windows 11 operating system.

Results. Handwriting recognition software has been developed.

The relevance and problematic of the work is that the demand for machine learning in the IT industry goods market is growing as, at the same time, the digitalization of document flow and everyday transmission of text information.

Scientific Novelty of the Obtained Results:

- the methodology of preprocessing handwritten images has been improved by combining adaptive binarization, normalization, and contour filtering, which enhances the quality of input data before being fed into the deep learning model;
- a modified recursive neural network architecture is proposed, optimized for recognizing small handwritten characters by integrating convolutional blocks with layers that increase sensitivity to local image features;

- the training approach for the model has been improved through the use of an extended augmentation system (simulation of uneven pressure, variations in slant and noise), which increases the network's robustness to different types of handwriting;

- a software application has been developed using a custom machine learning model, ensuring the integration of preprocessing, recognition, and visualization processes into a unified system.

Conclusions. Software has been designed and implemented for handwriting recognition, and the developed software has been tested.

The field of use is the application of the software in many areas, in particular for organizing the educational process in educational institutions, digitizing archival documents of state enterprises, and so on.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	10
Вступ.....	11
1 Аналіз предметної області.....	13
1.1 Огляд існуючих методів вирішення завдання.....	13
1.2 Огляд існуючих програмних засобів, що вирішують аналогічні завдання.....	14
1.2.1 Програмний продукт Transkribus.....	14
1.2.2 Програмний продукт Google Cloud Vision	15
1.2.3 Програмний продукт Tesseract.....	18
1.2.4 Порівняльна таблиця існуючих та розроблюваного програмних продуктів.....	19
1.3 Постановка завдання до роботи.....	21
1.4 Висновки за розділом 1.....	24
2 Обґрунтування вибору програмних засобів розробки	25
2.1 Вибір програмної платформи.....	25
2.1.1 Amazon Machine Learning.....	26
2.1.2 Azure Machine Learning Studio	27
2.1.3 TensorFlow.....	28
2.1.4 Порівняльна таблиця існуючих програмних платформ, призначених для реалізації задач машинного навчання.....	30
2.2 Python і його можливості.....	31
2.3 Вибір інтегрованого середовища розробки.....	34
2.4 Висновки за розділом 2.....	36
3 Основні рішення щодо реалізації компонентів системи.....	37
3.1 Вибір підходу.....	37
3.2 Алгоритм функціонування розробленого програмного забезпечення....	40
3.2.1 Метод бінарізації зображення.....	41
3.2.2 Метод сегментації тексту.....	44

3.2.3 Модифікований метод сегментації тексту.....	48
3.2.4 Тренування нейронної мережі.....	50
3.2.5 Метод перевірки слова у словнику.....	55
3.3 Опис модулів розроблюваної системи.....	56
3.4 Висновки за розділом 3.....	59
4 Експлуатація, тестування та експериментальне дослідження програми.....	60
4.1 Інструкція роботи користувача з програмою.....	60
4.2 Тестування програми.....	62
4.3 Висновки за розділом 4.....	69
Висновки.....	71
Перелік джерел посилань.....	73
Додаток А Фрагмент тексту програми.....	76
Додаток Б Слайди презентації.....	80

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application programming interface;
CNN	– Convolutional Neural Network;
CSS	– Cascading Style Sheets;
GUI	– Graphical user interface;
HTML	– HyperText Markup Language;
IDE	– Integrated Development Environment;
OCR	– Optical Character Recognition;
RNN	– Recurrent Neural Network;
ПЗ	– програмне забезпечення;
ЦЗ	– цифрове зображення;
ШІ	– штучний інтелект.

ВСТУП

Писемність вже давно є невід’ємною частиною не лише людської культури, а й зберігання різноманітної повсякденної інформації людьми. І якщо писемні знахідки цікавлять багатьма факторами як артефакт втраченої культури, то наша повсякденна текстова інформація в першу чергу цікава саме як інформація. А це змінює підходи до її збереження. Навіщо накопичувати цілі архіви старих документів, якщо можна зберегти їх у цифровому архіві? Навіщо переписувати руками важливий для вас рецепт чи розділ у підручнику, якщо можна скопіювати в окремий файл із помітками та відкритим доступом з будь-якого вашого пристрою?

Саме через зручність збереження інформації в цифровому вигляді, ідея перетворення рукописного текстового запису на цифровий файл є актуальною, а поліпшення засобів для цього є доцільним.

В останні роки системи розпізнавання друкованого тексту стали досить ефективними. Рукописні цифри та інтерактивне написання розпізнаються якісно. Однак поточні технології все ще знаходяться на обмеженому рівні, щоб розпізнавати текстові зображення різних стилів та мов рукописного вводу [1]. Якість розпізнавання сильно залежить від мови, якості зображення, почерку людини що писала. До того ж, існуючі системи спрямовані на більш широкі задачі аніж розпізнавання тексту. Деякі спеціалізуються на розпізнаванні історичних текстів, друкованих текстів, деякі на розпізнаванні будь-яких об’єктів на зображенні, лише одним з яких, є текст [2].

Отже, дана кваліфікаційна робота спрямована на розроблення та вдосконалення підходу до системи розпізнавання рукописного тексту, виходячи з принципу, що якісна попередня обробка є запорукою успішного розпізнавання. У межах етапу попередньої обробки підхід охоплює приведення зображення до бінарного вигляду, визначення текстових областей, подальше опрацювання – зокрема корекцію фону, сегментацію рядків, розпізнавання символів і фільтрацію шумів.

Для досягнення мети необхідно вирішити наступні завдання:

- провести огляд літератури з питань розпізнавання рукописного тексту;
- визначити основні підходи до вирішення задачі;
- провести аналіз програмних засобів для розпізнавання рукописного тексту;
- спроектувати підхід до розпізнавання тексту, що включає найсильніші сторони існуючих провідних підходів;
- розробити програмний застосунок що використовує знайдений підхід.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд існуючих методів вирішення завдання

Завдання розпізнавання рукописного тексту, завдяки мінливості вхідних даних, можна назвати складним. І ця складність допомогла пройти моделям великий шлях розвитку від ранніх евристичних підходів до сучасних нейронних моделей, які використовують методи глибокого навчання. Сфера застосування цієї галузі також розширилася: моделі, спочатку здатні розпізнавати лише вміст на рівні слів, перейшли до нещодавніх наскрізних підходів на рівні документів [3].

Історично системи розпізнавання рукописного тексту пройшли кілька технологічних етапів розвитку. Перші підходи базувалися на евристичних і правила-орієнтованих методах, які використовували вручну обрані ознаки для розпізнавання символів і слів [4]-[8]. Такі системи мали обмежену здатність працювати з великою варіативністю почерку і часто потребували складного попереднього опрацювання та сегментації окремих символів [9]. Поява машинного навчання стала справжнім кроком уперед, відкривши шлях до створення більш гнучких і надійних систем розпізнавання. Статистичні методи, зокрема моделі прихованих марковських процесів [10], забезпечили помітне покращення, оскільки навчилися розпізнавати цілі слова або рядки на основі розмічених даних [11]-[13].

Виникнення глибокого навчання [14] справило революцію у сфері систем розпізнавання, забезпечивши безпрецедентний прогрес у точності розпізнавання. Згорткові нейронні мережі (CNN) [15] та рекурентні нейронні мережі (RNN) [16] показали чудові результати у виявленні просторових і послідовних залежностей у рукописному тексті. Вибір типу нейронної мережі проведемо ближче до 3 розділу і там же подивимося їх особливості.

На сьогодні, можна знайти багато сфер, де нейронні мережі знайшли широке застосування, хоча й не несуть практичного покращення. В сфері розпізнавання рукописного тексту, таке твердження є хибним.

Окрім підвищення точності, такі технологічні зрушення також розширили підходи до розпізнавання рукописного тексту. Якщо раніше системи були зосереджені на розпізнаванні окремих слів чи рядків, то сучасні методи вже дозволяють відтворювати зміст цілих абзаців або навіть документів. Далі розберемо декілька прикладів втілення цих методів.

1.2 Огляд існуючих програмних засобів, що вирішують аналогічні завдання

1.2.1 Програмний продукт Transkribus

Однією з провідних сучасних платформ для автоматизованого розпізнавання текстів історичних документів є Transkribus. Розпізнавання історичних текстів дуже близько підходить до теми роботи, що розглядається, адже історичні тексти рукописні, а якість матеріалу, на якому текст міститься, може спричинити створення прородніх шумів на зображенні, що ускладнить розпізнавання і наблизить задачі систем, які ми розглядаємо. Transkribus забезпечує комплексну обробку документів, включно з розпізнаванням рукописного та друкованого тексту, аналізом зображень і визначенням структурних елементів сторінок (заголовків, колонок, підписів тощо).

Transkribus об'єднує рішення, розроблені провідними науковими групами Європи, серед яких Pattern Recognition and Human Language Technology (PRHLT) Технічного університету Валенсії та Computational Intelligence Technology Lab (CITlab) Університету Ростока. Така інтеграція забезпечує високий рівень точності розпізнавання, можливість навчання моделей на конкретних типах рукописів і гнучкість у застосуванні до різних мов та історичних періодів [17]. Проте, коректне зчитування з української мови мені отримати не вдалося, хоча підтримка української мови і значиться в можливостях системи. Результат спроби використання Transkribus без додаткового налаштування можна побачити на рис. 1.1.

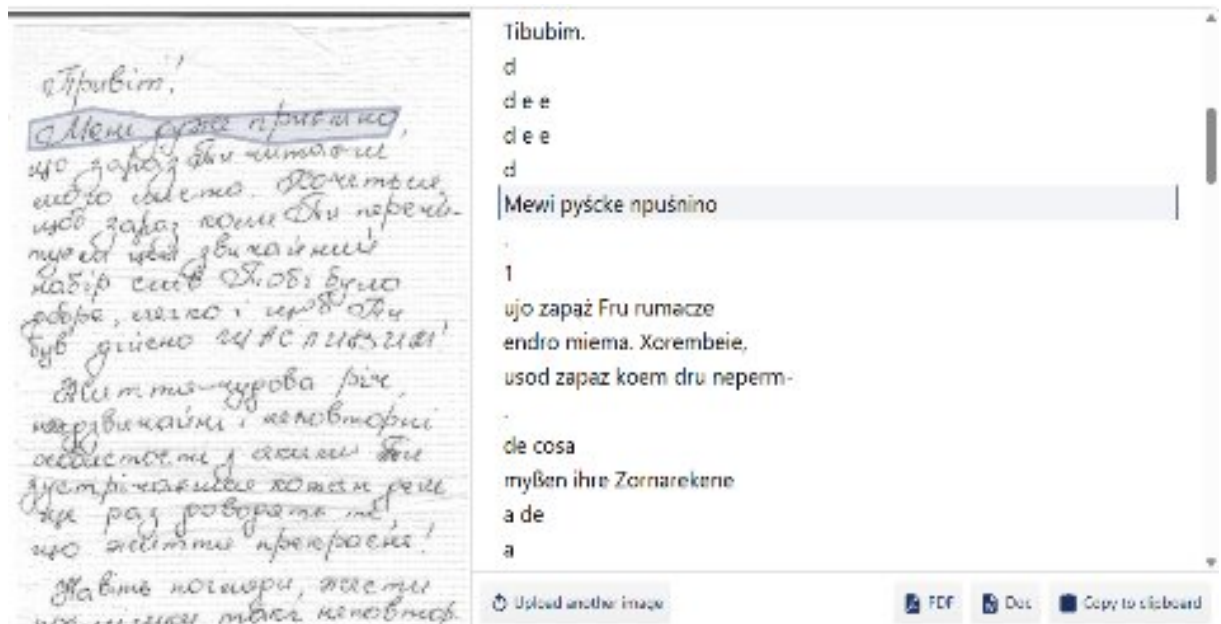


Рисунок 1.1 – Результат роботи Transkribus

Може здатися, що Transkribus не впорався навіть із визначенням мови, проте це міжнародна демо-версія для перегляду загальних можливостей системи і попри це, визначення тексту на зображенні, поділ його на рядки та графічний інтерфейс роботи з текстом виконані на найвищому рівні. А точність роботи з українською мовою, після налаштувань та додаткових тренувань, буде розглянута у розділі 4, де буде протестовано розроблений додаток та порівняно результати із аналогами, які роздивляємося зараз.

1.2.2 Програмний продукт Google Cloud Vision

Google Cloud Vision – це одна з найрозвиненіших комерційних платформ для аналізу та розпізнавання зображень, розроблена компанією Google у межах екосистеми Google Cloud Platform. Сервіс використовує алгоритми глибинного навчання та нейронні мережі, що забезпечують автоматичне виявлення, класифікацію та інтерпретацію об'єктів на зображеннях [18].

Розпізнавання тексту є лише однією з багатьох можливостей даної платформи, та необхідні для тематики роботи функції, вона включає.

Основними функціональними можливостями Google Cloud Vision є:

- оптичне розпізнавання тексту (OCR) з друкованих та рукописних документів;
- ідентифікація об'єктів, осіб, логотипів, місць і сцен;
- розпізнавання мови тексту та визначення його контексту;
- аналіз емоцій на обличчях, виявлення небажаного або чутливого контенту;
- класифікація зображень за категоріями (наприклад, природа, архітектура, транспорт тощо).

Перевагою платформи є її висока точність та масштабованість, оскільки всі обчислення виконуються у хмарному середовищі Google. Це дозволяє обробляти великі обсяги даних і швидко інтегрувати технологію у вебсервіси, мобільні застосунки та корпоративні рішення через інтерфейс програмування додатків (API).

Універсальність та потужність алгоритмів сервісу роблять його ефективним засобом для загального розпізнавання тексту й аналізу візуального контенту не дивлячись на більш широку спеціалізацію ніж лише зчитування тексту. У поєднанні з іншими сервісами Google Cloud (наприклад, AutoML Vision або Document AI), система може навіть забезпечити повноцінну автоматизацію процесів цифрової обробки документів. Проте, точність зчитування рукописного тексту українською, можна побачити на рис. 1.2.

Оскільки Google Cloud Vision – не програмний засіб, а платформа, існує кілька варіантів звернення до її можливостей. Серед способів застосування технології можна виділити:

- через REST API (HTTP-запити);
- через клієнтські бібліотеки;
- через Google Cloud Console;
- через готові сервіси та інструменти.

1.2.3 Програмний продукт Tesseract

Tesseract – це одна з найвідоміших і найпоширеніших систем оптичного розпізнавання тексту (OCR) з відкритим вихідним кодом. Її первинна розробка розпочалася у 1980-х роках у лабораторіях компанії Hewlett-Packard (HP), а згодом, у 2005 році, проєкт був переданий у відкритий доступ і підтримується Google [19].

Система реалізує двохетапний підхід до розпізнавання тексту: спочатку відбувається сегментація зображення на символи, слова та рядки, після чого застосовуються рекурентні нейронні мережі (зокрема, архітектура LSTM – Long Short-Term Memory) для інтерпретації текстового вмісту. Такий підхід дозволив суттєво підвищити точність розпізнавання, особливо для різних мов та типів шрифтів.

Основні характеристики Tesseract:

- підтримка понад 100 мов, включно з українською;
- можливість донавчання моделей для спеціалізованих шрифтів або рукописів;
- повна відкритість вихідного коду та кросплатформність (Linux, Windows, macOS);
- сумісність з іншими бібліотеками, зокрема OpenCV, що дає змогу виконувати попередню обробку зображень (очищення, контрастування, корекція нахилу тощо).

Tesseract часто використовується як базовий OCR-движок у багатьох комерційних і наукових рішеннях. Попри те, що він не має настільки гнучкого інтерфейсу чи розширених можливостей машинного навчання, як Google Cloud Vision або Transkribus, його безкоштовність, відкритість і можливість кастомізації роблять Tesseract оптимальним вибором для дослідницьких та освітніх проєктів, а також для інтеграції у власні програмні системи. Результат роботи зображено на рис. 1.3.

розпізнавання не лише тексту, а будь-яких об'єктів на зображеннях, а Tesseract як API для розпізнавання саме тексту.

Проте, всі 3 засоби, без додаткових налаштувань, не змогли впоратися із листом, який я хочу використати як фінальне випробування для програмного продукту, який стане результатом даної роботи. Деякі навіть не розпізнали мову, або зробили це навірно. Не дивлячись на це, кожен засіб показав потенціал у виконанні цього завдання і у розділі 4, де буде протестовано розроблений продукт, результати тестування можуть сильно відрізнятись через додаткові налаштування систем.

Тож, результат даної роботи повинен підтримувати такі особливості: українська мова, новіші алгоритми, відсутність потреби у завантаженні, менш широка область застосування і масштабованість, сучасний та інтуїтивно-зрозумілий інтерфейс.

Може здатися, що менш широка область застосування – це погіршення, проте потреба в додаткових налаштуваннях виходить саме із ширшого кола задач кожного із продуктів. Тож, такий пункт дозволяє позбутися необхідності у навчанні користувача для роботи з програмою.

В таблиці 1.1 проведено порівняння за деякими показниками вже існуючих програмних продуктів та програми, яка буде розроблена в ході виконання роботи.

Таблиця 1.1 – Порівняння програмних продуктів

Критерій	Transkribus	Google Cloud Vision	Tesseract OCR	Створений проєкт
1	2	3	4	5
Тип системи	Спеціалізована платформа для історичних документів	Хмарний сервіс загального призначення	Відкрита бібліотека OCR з відкритим кодом	Невеликий застосунок вузького призначення
Доступність	Обмежений безкоштовний доступ; комерційна підписка	Хмарний сервіс із платною моделлю (оплата за запити)	Повністю безкоштовна та відкрита	Повністю безкоштовна та відкрита

Продовження таблиці 1.1

1	2	3	4	5
Точність розпізнавання	Висока (до 98% при навчанні на конкретному наборі даних)	Висока для друкованих текстів ($\approx 95\text{--}98\%$)	Середня/висока ($\approx 90\text{--}95\%$, залежно від якості зображення)	Середня/висока ($\approx 90\text{--}95\%$, залежно від якості зображення)
Переваги	Висока точність для історичних рукописів, навчання моделей	Потужна інфраструктура, розпізнавання об'єктів	Безкоштовність, відкритий код, гнучкість, кросплатформність	Підтримка української, легкість використання
Недоліки	Необхідність навчання моделей, обмежений безкоштовний доступ	Платна модель, відсутність відкритого коду	Менша точність для рукописів, потреба в попередній обробці зображень	Невеликий функціонал в порівнянні з розглянутими засобами

1.3 Постановка завдання до роботи

Задача розпізнавання рукописного тексту належить до класу задач класифікації образів та послідовного розпізнавання.

Нехай задано множину вхідних зображень $X = \{x_1, x_2, \dots, x_N\}$, де кожне зображення $x_i \in R^{m \times n}$ – це матриця піксельних інтенсивностей розміру $m \times n$, а відповідні правильні мітки класів – це символи або послідовності символів $Y = \{y_1, y_2, \dots, y_N\}$, де $y_i \in A^*$, а A – алфавіт допустимих символів (наприклад, цифри 0 – 9 або літери латиниці/кирилиці).

Потрібно знайти відображення, зображене на формулі (1.1):

$$f_{\theta}: \{R\}^{m \times n} \rightarrow A^*. \quad (1.1)$$

Математичний вигляд завдання використовує наближення замість рівності через неможливість отримання гарантованого результату.

Знайдене відображення, параметризоване вектором параметрів θ , яке мінімізує функціонал помилки (втрат) на тренувальній вибірці (X, Y) знайдемо у формулі (1.2):

$$\theta^* = \arg \min_{\theta} \frac{1}{N} \sum_{i=1}^N L(f_{\theta}(x_i), y_i), \quad (1.2)$$

де $L(\cdot, \cdot)$ – функція втрат, що вимірює відмінність між передбаченим результатом і правильним класом.

Та розгляд цілого слова потребує надто багато ресурсів на обчислення, тож трохи змінимо постановку завдання.

У випадку, коли $y_i \in A$ (один символ), задача є мультикласовою класифікацією, де класи представляють собою алфавіт, до якого віднесемо не лише літери, а й усі можливі допустимі для розпізнавання символи. На пошук класу потрібно накласти обмеження.

Нехай вихід моделі – це вектор ймовірностей розподілу символів по класам(алфавіту), заданий формулою (1.3):

$$\hat{p}(y | x_i) = f_{\theta}(x_i), \quad (1.3)$$

де $\hat{p}(y | x_i) \in [0,1]^{|A|}$ і $\sum_{y \in A} \hat{p}(y | x_i) = 1$.

Тобто, кожен символ точно є в алфавіті і належить лише одному класу. Такі припущення можна буде легко реалізувати при передобробці зображення і при програмуванні, та це дозволяє назвати задачу вирішуваною. Та нам потрібна не просто ймовірність розподілу, а спосіб визначення класу, що зазвичай проводиться через пошук критерію.

Оптимізація пошуку ймовірності проводиться за критерієм функції втрат L , зображеним на формулі (1.4):

$$L = -\frac{1}{N} \sum_{i=1}^N \log \hat{p}(y_i | x_i). \quad (1.4)$$

У цьому випадку для кожного зображення x_i необхідно визначити послідовність символів, що можна зобразити формулою (1.5):

$$y_i = (y_{i1}, y_{i2}, \dots, y_{iT_i}), \quad (1.5)$$

де T_i – довжина послідовності.

Тоді відображення f_θ будується як послідовність умовних розподілів для кожного вхідного символу, що математично зобразимо формулою (1.6):

$$\hat{p}(y_i | x_i) = \prod_{t=1}^{T_i} \hat{p}(y_{it} | x_i, y_{i1}, \dots, y_{i(t-1)}). \quad (1.6)$$

Для навчання таких моделей часто використовується функція втрат CTC (Connectionist Temporal Classification – це функція втрат для навчання глибоких нейронних мереж, що вирішує проблеми з послідовностями, де часова синхронізація може бути нерівномірною [20]. Її основна ідея полягає в тому, щоб усунути необхідність у заздалегідь вирівняних даних, що підходить до нашої теми), що дозволяє зіставляти змінні довжини вхідних та вихідних послідовностей. Математичне представлення функції втрат виділено у формулі (1.7):

$$L_{CTC} = - \sum_{(x_i, y_i)} \log P(y_i | x_i; \theta). \quad (1.7)$$

Така функція втрат набагато краще відповідає поставленій задачі, проте навчання нейронної мережі потребує додавання нових обмежень та вимог для зменшення використання ресурсів. Це дозволяє виділити такі вимоги:

- вхідні зображення повинні бути нормалізовані за розміром, контрастом та яскравістю;

- вибір моделі f_{θ} повинен забезпечувати інваріантність до афінних перетворень рукописних символів (зміщення, масштаб, нахил);
- похибка розпізнавання повинна задовольняти умову точності, виражену формулою (1.8):

$$Acc(f_{\theta}) = \frac{1}{N} \sum_{i=1}^N 1\{f_{\theta}(x_i) = y_i\} \geq \alpha, \quad (1.8)$$

де α – заданий поріг точності (наприклад, $\alpha \geq 0.95$).

Метою розділу є побудова математичної моделі для подальшої алгоритмічної реалізації програмного засобу для розпізнавання рукописного тексту на основі методів машинного навчання.

Для досягнення мети необхідно:

- провести математичний аналіз існуючих методів розпізнавання та визначити придатну модель f_{θ} ;
- розробити алгоритм навчання параметрів θ на тренувальній вибірці;
- реалізувати та експериментально дослідити точність і швидкість алгоритму;
- оцінити вплив розміру вибірки, архітектури мережі та параметрів регуляризації на функціонал втрат $L(\theta)$.

1.4 Висновки за розділом 1

В першому розділі було проаналізовано предметну область, її розвиток та вже існуючі програмні аналоги проєкту.

У ході аналізу було виявлено переваги та недоліки аналогів, функції, що повинен задовольняти проєкт. Актуальність проєкту лише підвищилася завдяки знаходженню вагомих недоліків в роботі аналогів.

Як результат проведених досліджень, було сформульовано завдання на проєкт, розглянуто його математичну модель і сформовані кроки для досягнення мети.

2 ОБҐРУНТУВАННЯ ВИБОРУ ПРОГРАМНИХ ЗАСОБІВ РОЗРОБКИ

2.1 Вибір програмної платформи

Останніми роками машинне навчання стало однією з найпотужніших тенденцій у сфері інформаційних технологій. Цей напрям активно розвивається завдяки не лише здешевленню хмарних обчислювальних ресурсів, але й широкій доступності високопродуктивних процесорів та графічних карт, що використовуються для обчислень такого типу. Крім того, з'явилася значна кількість фреймворків для реалізації алгоритмів машинного навчання. Більшість із них мають відкритий вихідний код, проте найважливішим є те, що ці фреймворки спроектовані таким чином, щоб спростити роботу з найскладнішими аспектами машинного навчання, роблячи ці технології більш доступними для широкого кола розробників.

Існує три основні причини, через які виникає потреба у використанні машинного навчання, коли традиційне програмування є недостатнім. По-перше, розробники не здатні передбачити всі можливі ситуації, у яких може опинитися система. Наприклад, робот, призначений для проходження лабіринтів, повинен самостійно вивчати план кожного нового лабіринту, з яким він стикається. По-друге, неможливо врахувати всі зміни, що відбуваються з часом. Так, програма для прогнозування біржових цін має навчатися та адаптуватися до змін ринкових умов. По-третє, у деяких випадках люди просто не знають, як запрограмувати вирішення певної задачі. Наприклад, більшість людей без зусиль розпізнають обличчя знайомих, однак навіть досвідчені програмісти не можуть створити ефективну програму для цього без використання методів машинного навчання.

Розглянемо аналіз сучасних платформ, призначених для реалізації задач машинного навчання.

2.1.1 Amazon Machine Learning

Сервіс Amazon Machine Learning (рис. 2.1) пропонує користувачам зручні інструменти візуалізації, які покроково супроводжують процес створення моделей, не вимагаючи глибоких знань у сфері складних алгоритмів чи технологій машинного навчання. Після побудови моделі цей вебсервіс дозволяє легко отримувати передбачення для практичного застосування за допомогою простого прикладного програмного інтерфейсу (API) [21].

Сервіс підключається до даних, що зберігаються в хмарних базах даних або сховищах, і може виконувати завдання двійкової класифікації, багатокласової категоризації, а також регресійного аналізу на основі наданих даних для побудови моделі.

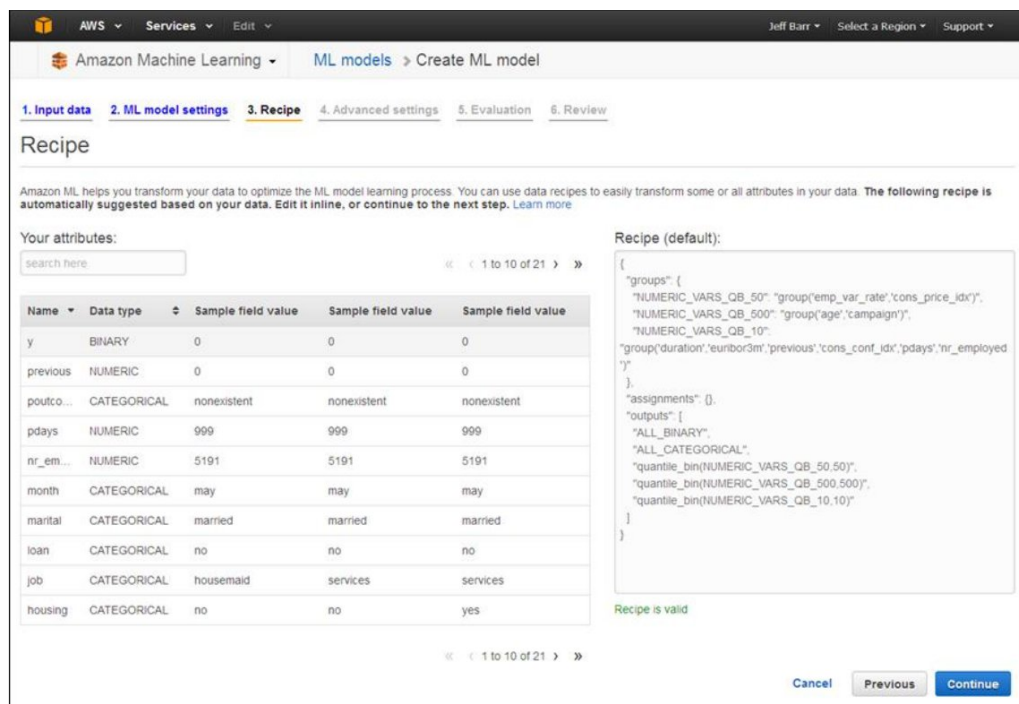


Рисунок 2.1 – Інтерфейс сервісу машинного навчання від Amazon

Варто зазначити, що сервіс працює виключно з даними, розміщеними у хмарних сховищах компанії, а імпорт чи експорт моделей неможливий. Крім того, обсяг навчальних вибірок обмежений розміром у 100 ГБ.

2.1.2 Azure Machine Learning Studio

Майкрософтівська студія машинного навчання – Azure Machine Learning Studio.

З огляду на значні обсяги даних і високі обчислювальні вимоги, характерні для задач машинного навчання, хмарні платформи є оптимальним середовищем для реалізації ML-додатків. Компанія Microsoft інтегрувала у свій хмарний сервіс, який потребує підписки, Azure власну платформу машинного навчання, за використання якої можна сплачувати лише за фактичне споживання ресурсів – Azure Machine Learning Studio (рис. 2.2).

Azure Machine Learning Studio надає можливість створювати, тренувати та розгортати моделі машинного навчання, забезпечуючи їх інтеграцію з іншими сервісами через інтерфейси прикладного програмування. Користувачам надається до 10 ГБ простору для зберігання даних на один обліковий запис, однак можна підключати й власні сховища Azure Storage.

Платформа підтримує широкий набір алгоритмів, розроблених як Microsoft, так і сторонніми розробниками. Для ознайомлення з функціоналом сервісу навіть не потрібно створювати обліковий запис – користувач може увійти анонімно та працювати зі студією протягом восьми годин у тестовому режимі [22].

Сервіс Microsoft Azure Machine Learning є частиною Cortana Intelligence Suite, що дозволяє взаємодію з даними з використанням природної мови і усно за допомогою Cortana.

На додаток до технології .NET при створенні програм для Azure можуть використовуватися такі технології, як Java, PHP, C / C++ або Python.

Проте, документація майкрософт, за роки, розвинулася настільки, що потребує своєї документації для орієнтування.

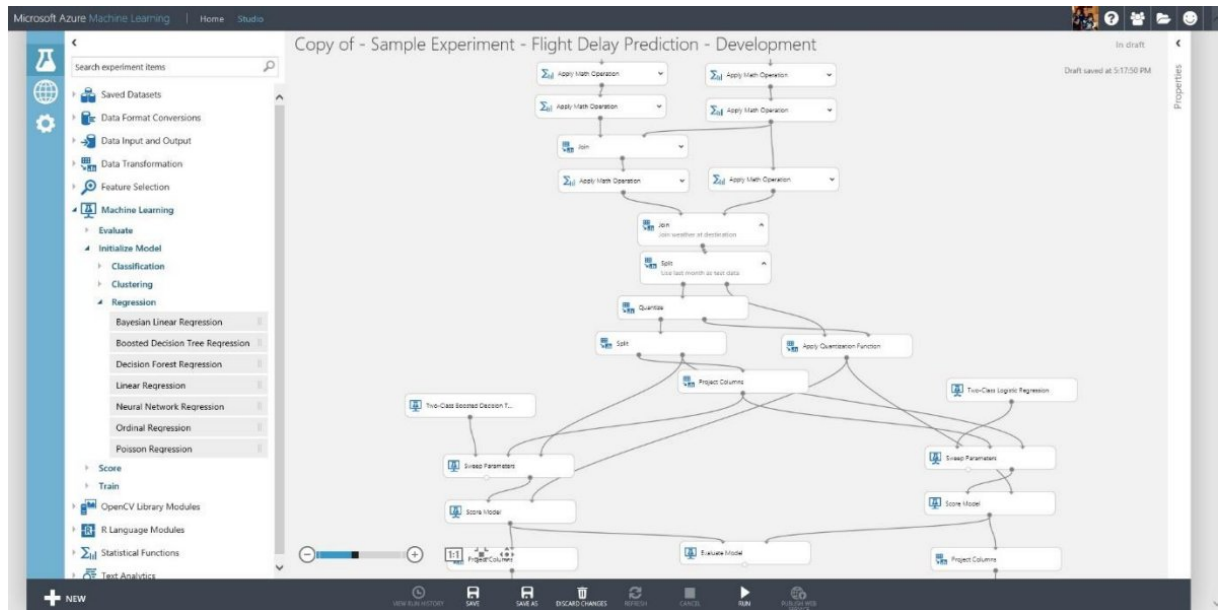


Рисунок 2.2 – Інтерфейс сервісу машинного навчання від Microsoft

Також можна згадати про існування Cognitive Services (раніше Project Oxford) – це набір API, SDK та сервісів, доступних для розробників, що дозволяють створювати програми з використанням інтелектуальних алгоритмів та машинного навчання, та розглядати будемо лише якщо не знайдеться варіанту краще.

2.1.3 TensorFlow

TensorFlow (рис. 2.3) від Google – це потужний фреймворк для машинного навчання, розроблений із метою забезпечення розподілених обчислень у межах кластерів. Спочатку він створювався для вирішення внутрішніх завдань компанії Google, проте згодом був опублікований як продукт із відкритим вихідним кодом, що дало змогу широкій спільноті розробників долучитися до його вдосконалення.

TensorFlow реалізує концепцію графів потоків даних, у яких порції даних (так звані тензори) передаються між вузлами графа для послідовної обробки за допомогою набору алгоритмів. Переміщення даних між обчислювальними вузлами відбувається у вигляді «потоків».

Створення та керування графами можливе за допомогою багатьох мов, таких як C++ або Python, а виконання обчислень може здійснюватися як на центральних процесорах (CPU), так і на графічних процесорах (GPU), що забезпечує високу продуктивність і масштабованість [23].

Компанія Google має довгострокові плани щодо подальшого розвитку TensorFlow спільними зусиллями з незалежними розробниками, сприяючи розширенню його функціональних можливостей і застосуванню у найрізноманітніших галузях.

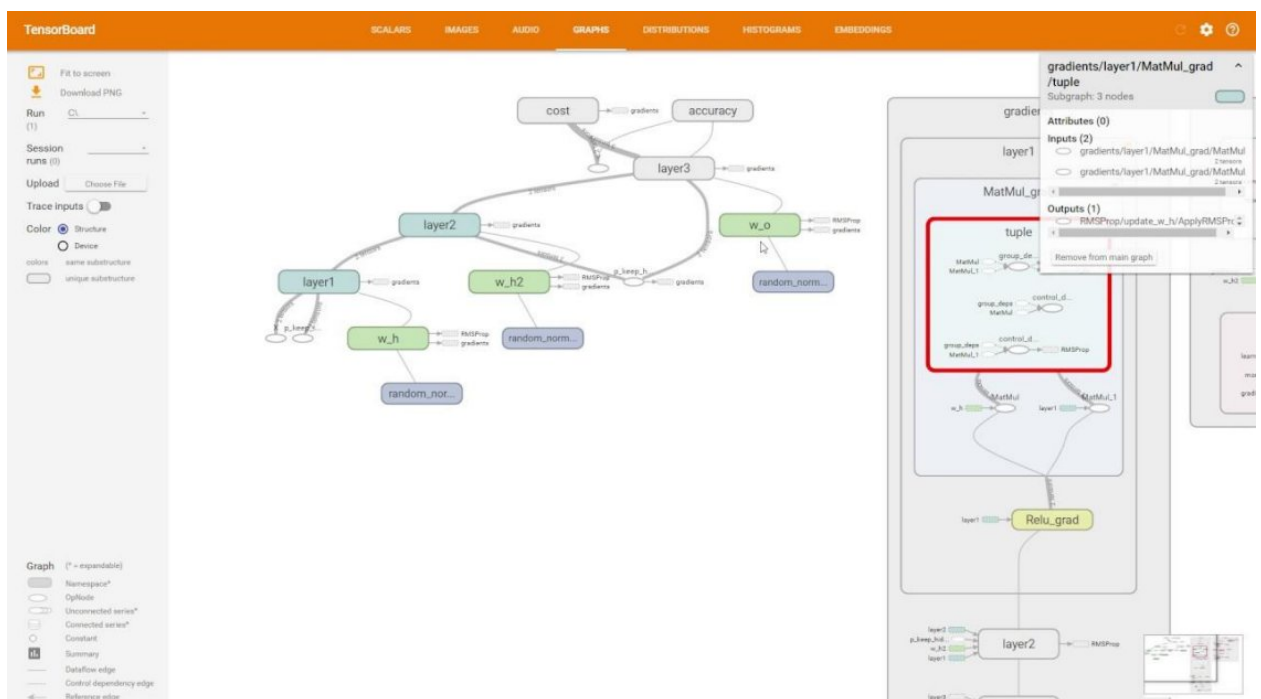


Рисунок 2.3 – Інтерфейс сервісу машинного навчання від Google

TensorFlow має інтегровану високорівневу бібліотеку Keras, яка спрощує створення, тренування та оцінювання нейронних мереж, роблячи розробку моделі інтуїтивно зрозумілою. А завдяки TensorFlow.js, модель розпізнавання рукописного тексту можна розгорнути на мобільних пристроях або безпосередньо у веббраузері, що робить її зручною для практичного застосування.

Отже, TensorFlow – це оптимальний вибір для розробки системи розпізнавання рукописного тексту, оскільки він поєднує гнучкість, високу

продуктивність, зручність у розробці та потужну підтримку спільноти, що забезпечує швидке створення точної та ефективної моделі.

Для ефективного використання можливостей TensorFlow важливо обрати відповідну мову програмування, яка забезпечить зручність у розробці, налагодженні та експериментуванні з моделями машинного навчання.

TensorFlow підтримує кілька мов (зокрема C++, JavaScript, Java), проте основною і найзручнішою мовою для роботи з цим фреймворком є Python.

2.1.4 Порівняльна таблиця існуючих програмних платформ, призначених для реалізації задач машинного навчання

Не дивлячись на те, що в огляді платформ вже можна побачити явну підтримку автором однієї із технологій, порівняльна таблиця повинна наглядно продемонструвати плюси і мінуси усіх платформ і підтвердити чи спростувати зроблений вибір. Для оформлення порівняльної таблиці 2.1, підсумуємо інформацію з трьох попередніх підрозділів.

Таблиця 2.1 – Порівняння існуючих програмних платформ, призначених для реалізації задач машинного навчання

Критерій	Amazon Machine Learning	Azure Machine Learning Studio	TensorFlow
1	2	3	4
Тип платформи	Хмарний сервіс машинного навчання	Хмарна та локальна платформа ML з інтегрованим середовищем	Фреймворк з відкритим кодом для глибокого навчання
Гнучкість і кастомізація	Обмежена	Середня (блокові сценарії та можливість власного коду)	Дуже висока (повна свобода розробника)
Продуктивність	Залежить від AWS-ресурсів, оптимізовано під стандартні ML-задачі	Висока, особливо з використанням Azure GPU/TPU	Максимальна при використанні GPU/TPU; оптимізовано для глибокого навчання

Продовження таблиці 2.1

1	2	3	4
Можливості моделювання	Підтримка базових моделей (класифікація, регресія, рекомендаційні моделі)	Широкий перелік алгоритмів, включаючи класичні ML та моделі глибокого навчання	Глибоке навчання, гнучке створення складних нейронних мереж
Вартість	Оплата за використання ресурсів AWS, окремі тарифи за навчання моделей	Оплата за обчислювальні ресурси Azure	Безкоштовний інструмент; витрати лише на обладнання або хмарні ресурси
Призначення	Швидке розгортання простих ML-моделей	Побудова, тестування і деплой комплексних ML-рішень	Розробка та навчання кастомних моделей глибокого навчання
Переваги	Простота використання, інтеграція з AWS	Гнучкість, візуальне моделювання, готові компоненти	Потужність, підтримка складних моделей
Недоліки	Обмежені моделі, невисока гнучкість	Потребує підписки Azure, інколи складні налаштування	Складний у вивченні

Як видно з таблиці, TensorFlow повністю підходить під поставлені задачі. Найбільший вплив на вибір мали призначення, можливості моделювання і продуктивність платформи і по всіх показниках TensorFlow має найбільш відповідні показники. Також, приємним бонусом є вартість. А переваги і недоліки являють собою стислий підсумок сильних і слабких сторін, зроблений при міркуванні над вибором.

2.2 Python і його можливості

Python має низку переваг, які роблять його ідеальним вибором для створення програми з розпізнавання рукописного тексту:

- тісна інтеграція з TensorFlow. TensorFlow спочатку розроблявся саме для Python, тому більшість його функцій, бібліотек і прикладів

реалізовані саме цією мовою. Це забезпечує максимальну сумісність і простоту використання;

- широка екосистема бібліотек для обробки даних. Python підтримує потужні інструменти, такі як NumPy, Pandas, OpenCV, Matplotlib, Scikit-learn, які спрощують обробку, візуалізацію та аналіз даних перед навчанням моделі;

- простота синтаксису та швидкість розробки. Мова має зрозумілий синтаксис, що дозволяє розробникам зосередитися на логіці алгоритмів, а не на технічних деталях реалізації;

- підтримка великої спільноти. Python є найпопулярнішою мовою серед дослідників штучного інтелекту. Це означає доступ до великої кількості прикладів, навчальних матеріалів та активної спільноти розробників, готових допомогти у вирішенні проблем;

- зручність розгортання моделей. Завдяки Python можна легко інтегрувати модель TensorFlow у вебсервіси, десктопні чи мобільні застосунки за допомогою фреймворків на кшталт Flask, FastAPI або TensorFlow Lite.

Отже, вибір мови Python для реалізації програми розпізнавання рукописного тексту на базі TensorFlow є логічним і технічно обґрунтованим рішенням, яке забезпечує оптимальне поєднання продуктивності, гнучкості та зручності розробки.

Однак, попри всі переваги, Python має певні обмеження у створенні сучасного та привабливого графічного інтерфейсу користувача. Інструменти на кшталт Tkinter чи PyQt є функціональними, але не завжди відповідають сучасним вимогам до дизайну, зручності та адаптивності. Саме тому для реалізації інтерфейсу користувача буде використано вебтехнології – HTML, CSS та JavaScript. Вони дозволять створити сучасний, інтерактивний та зручний інтерфейс, який можна запускати у будь-якому браузері або інтегрувати як вебчастину програми.

Отже, основна логіка машинного навчання та обробки даних буде реалізована мовою Python із використанням TensorFlow, а візуальна частина програми – засобами HTML, CSS та JavaScript, що забезпечить поєднання потужності штучного інтелекту з сучасним, зручним інтерфейсом.

Окрім зручної інтеграції з TensorFlow, Python має багато сильних сторін. Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять цю мову програмування привабливою для швидкої розробки програм, а також як засіб поєднування наявних компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій, так і у вихідній формі на всіх основних платформах. В мові програмування Python підтримується кілька парадигм програмування, зокрема: об'єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована.

Серед основних її переваг можна назвати такі:

- переносність програм (що властиве більшості інтерпретованих мов);
- чистий синтаксис (для виділення блоків слід використовувати відступи);
- стандартний дистрибутив має велику кількість корисних модулів (включно з модулем для розробки графічного інтерфейсу);
- можливість використання Python в діалоговому режимі (дуже корисне для експериментування та розв'язання простих задач);
- стандартний дистрибутив має просте, але разом із тим досить потужне середовище розробки, яке зветься IDLE і яке написане мовою Python;
- зручний для розв'язання математичних проблем (має засоби роботи з комплексними числами, може оперувати з цілими числами довільної величини, у діалоговому режимі може використовуватися як потужний калькулятор);

– відкритий код (можливість редагувати його іншими користувачами).

Python має ефективні структури даних високого рівня та простий, але ефективний підхід до об'єктно-орієнтованого програмування. Елегантний синтаксис Python, динамічна обробка типів, а також те, що це інтерпретована мова, роблять її ідеальною для написання скриптів та швидкої розробки прикладних програм у багатьох галузях на більшості платформ. Додаємо інтеграцію TensorFlow, практичний досвід автора проєкту у використанні мови, легкість підключення сучасного інтерфейсу, створеного з використанням веб-технологій і кращого варіанту не знайти.

2.3 Вибір інтегрованого середовища розробки

Серед популярних середовищ програмування, які можуть бути використані для розробки проєктів на Python, можна розглянути такі:

– PyCharm – це інтегроване середовище розробки, спеціалізоване для мови Python, створене компанією JetBrains. Воно має розширену підтримку фреймворків машинного навчання, інтелектуальну підсвітку синтаксису, автоматичне завершення коду, засоби налагодження та вбудовану систему керування пакетами. PyCharm зручний для великих проєктів, проте вимагає значних системних ресурсів, а повна функціональність доступна лише у платній версії (Professional Edition);

– Jupyter Notebook є веборієнтованим середовищем для створення інтерактивних документів, що поєднують код, текст і візуалізації. Це середовище широко використовується у наукових дослідженнях і для створення бази знань машинного навчання, оскільки дозволяє виконувати код поетапно та миттєво переглядати результати. Проте Jupyter не є повноцінним середовищем розробки програмного забезпечення – воно більше підходить для експериментів, тестування моделей і демонстрації результатів;

– Spyder – це ще одне середовище, орієнтоване на наукові обчислення та аналіз даних. Воно має схожий інтерфейс із MATLAB і добре інтегрується з бібліотеками NumPy, Pandas і Matplotlib. Попри це, можливості Spyder для роботи з вебтехнологіями чи масштабними проєктами є обмеженими, тож не підходить нам за призначенням;

– Visual Studio Code – це безкоштовне, кросплатформне та розширюване середовище розробки, створене компанією Microsoft. Воно підтримує широкий спектр мов програмування, включно з Python, JavaScript, HTML і CSS, що є надзвичайно важливим для даного проєкту, який поєднує елементи машинного навчання та вебінтерфейсу. Широка масштабованість функціоналу налаштовується через різноманітні розширення, доступні безкоштовно у каталозі всередині середовища.

Для швидкого порівняння наведених середовищ, побудуємо і розглянемо таблицю 2.2.

Таблиця 2.2 – Порівняння середовищ розробки

Критерій	PyCharm	Jupyter Notebook	Spyder	Visual Studio Code
Найкраще підходить для	Великих Python-проєктів	Наукових експериментів, швидких прототипів	Data Science і аналізу даних	Будь-яких проєктів: від ML до вебу, найгнучкіший вибір
Зручність для ML та Data Science	Непогана інтеграція з Python-інструментами	Дуже зручний для аналітики та експериментів	Зосереджений на наукових задачах	Висока інтеграція з Jupyter, Python, TensorFlow тощо
Переваги	Професійне середовище для Python	Чудове для аналітики та експериментів	Зручно для дослідників	Найбільш універсальне, легке, розширюване та продуктивне
Недоліки	Вимогливість до ресурсів	Не підходить для великих проєктів	Обмежена розширюваність	Потребує встановлення розширень, інколи складні налаштування

Враховуючи проведений огляд, Visual Studio Code виглядає найкращим варіантом завдяки гнучкості налаштувань та універсальності для такого неоднорідного проєкту.

2.4 Висновки за розділом 2

У другому розділі було здійснено порівняльний аналіз та обґрунтований вибір основних інструментів для реалізації програмного продукту з розпізнавання рукописного тексту на основі технологій машинного навчання.

Зокрема, було розглянуто кілька варіантів платформ для навчання моделей штучного інтелекту, мов програмування і середовищ розробки.

Отже, у результаті проведеного аналізу для подальшої розробки програмного продукту було обрано:

- платформу машинного навчання – TensorFlow;
- основну мову програмування – Python (у поєднанні з HTML, CSS і JavaScript для створення інтерфейсу);
- середовище розробки – Visual Studio Code.

TensorFlow було обрано як найбільш відповідний поставленому завданню інструмент завдяки найрозвинутішій підтримці гнучкого навчання власних глибоких нейронних мереж.

Python було обрано завдяки широкій підтримці засобів збору, підготовки та аналізу великих потоків даних та підтримці TensorFlow, а також легкому інтегруванню сучасних засобів побудови інтерфейсу.

Visual Studio Code було обрано за універсальність, можливість налаштувань і легкість роботи з іншими обраними технологіями.

Зроблені висновки і підготовка дозволяють перейти до проєктування системи.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Вибір підходу

В першому розділі вже було зазначено про існування декількох підходів до розробки систем розпізнавання тексту. Зокрема, можна виділити: Optical Character Recognition (OCR), Recurrent Neural Networks (RNN) та Long Short-Term Memory (LSTM).

Оптичне розпізнавання символів (OCR) використовується для ідентифікації рукописного або друкованого тексту шляхом аналізу форми, структури та взаємного розташування символів.

Основні переваги технології OCR полягають у наступному:

- висока швидкість обробки та розпізнавання, особливо у випадку друкованого тексту;
- здатність працювати з різними стилями рукописного письма;
- простота використання та наявність готових інструментів, таких як Tesseract OCR, що спрощують процес розробки й інтеграції у програмні рішення.

Водночас OCR має певні недоліки:

- низька стійкість до некоректного або нерозбірливого почерку;
- зниження точності при розпізнаванні текстів зі складними з'єднаннями чи накладанням символів;
- обмежена точність у порівнянні з розпізнаванням друкованого тексту.

Рекурентні нейронні мережі (RNN) – це тип штучних нейронних мереж, призначений для обробки послідовних даних, зокрема рукописного тексту.

Переваги RNN:

- здатність моделювати довготривалі залежності між символами у послідовностях;

- ефективність при розпізнаванні складних та нерегулярних рукописних зразків.

Недоліки RNN:

- значні обчислювальні витрати та тривалий процес навчання;
- чутливість до шуму та артефактів, що можуть виникати під час сканування;
- потреба у великих обсягах навчальних даних для досягнення високої точності.

Окремим різновидом рекурентних мереж є LSTM (Long Short-Term Memory) – удосконалена архітектура RNN, яка має підвищену здатність зберігати та керувати довготривалими залежностями в даних [24].

Переваги LSTM:

- ефективне моделювання контекстних зв'язків у рукописному тексті;
- здатність запам'ятовувати інформацію з великого діапазону символів, що забезпечує точніше розпізнавання.

Недоліки LSTM подібні до недоліків RNN і включають високу обчислювальну складність, чутливість до якості зображень та необхідність використання значних обсягів навчальних даних [25].

Таким чином, аналіз показує, що саме архітектура LSTM є найбільш доцільним та ефективним вибором для реалізації поставленої мети проєкту, оскільки вона поєднує гнучкість, здатність до навчання на послідовних даних і високу точність розпізнавання рукописного тексту у той час як OCR підходить лише для друкованого, а RNN саме на рукописному тексті покаже себе найгірше в порівнянні з LSTM.

Для доведення необхідності використання нейронної мережі, можна сформулювати задачу у математичному вигляді, що ми і зробимо.

Позначимо через $\mathcal{X}$ простір зображень, що містять рукописний текст. Кожен елемент $x \in \mathcal{X}$ представляється у вигляді тривимірного тензора в $R^{H \times W \times C}$, де H , W та C відповідають висоті, ширині та кількості каналів

зображення відповідно, і їхні розміри можуть відрізнятися між окремими прикладами.

Метою розпізнавання є отримання відповідного текстового представлення $y = (y_1, y_2, \dots, y_N)$, де $y_i \in \Sigma$ позначає окремий символ із заздалегідь визначеного алфавіту Σ . А алфавіт Σ може включати алфавітно-цифрові символи, розділові знаки або інші спеціальні токени, такі як пробіли чи символи переносу рядка.

І, з формальної точки зору, ця задача може бути представлена як задача статистичного відновлення за формулою (3.1):

$$y^* = \underset{y \in \Sigma^*}{\operatorname{argmax}} P(y | x), \quad (3.1)$$

де Σ^* – це множина всіх можливих послідовностей символів.

Відомо, що точне розв'язання рівняння (3.1) є обчислювально нездійсненним. Тож, ми можемо використовувати лише наближені методи та засоби що не залежать від однозначно вірної відповіді і це призводить до можливості модифікації методів вирішення задачі.

Частково це математично обґрунтування повторює математичну постановку завдання із першого розділу, та на цей раз сформулюємо більш конкретні вимоги.

У ході розробки, буде додано і розглянуто модифікації до кожного кроку роботи програми, проте план на переведення зображення у текст залишиться у вигляді:

- приведення зображення до оптимального для аналізу вигляду;
- відділення тексту від фону;
- поділ тексту на частини, які буде розпізнавати нейронна мережа;
- обробка нейронною мережею;
- форматування результату у звичний для кінцевого користувача вигляд.

3.2 Алгоритм функціонування розробленого програмного забезпечення

Основні методи, які використовуються при роботі із розробленою системою зображені на рис. 3.1.



Рисунок 3.1 – Функціональна схема розробленої системи

На схемі зображено потік даних по функціональним блокам програми. Перша функція викликається при завантаженні користувачем зображення у програму. Ця подія починає обробку зображення. Спочатку, бінарізація прибирає зайві кольори з зображення для полегшення сегментації. Наступним кроком, із зображення виділяється текст, із тексту рядки, із рядків слова, із слів символи. Спеціальні символи, такі як «,» або «+» вважаються окремими словами. Для розпізнавання, знайдені символи відправляються на

сервер, де з ними працює нейронна мережа. На клієнті зберігається відношення відправлених символів до конкретних слів а слів до рядків.

На сервері символи проходять повз 5 шарів, роботу яких буде описано пізніше. На виході, клієнту повертається ймовірнісний розподіл послідовності символів.

Після отримання вірогідних символів, на клієнті заповнюються порожні слоти у словах і слова перевіряються у словнику. Якщо слово знайдене у словнику, воно буде виведено в результаті. В іншому випадку, вважаємо що нейронна мережа знайшла символ у частині фону і слово пропускається. Сформований таким чином текст виводиться користувачу як результат роботи програми.

Як видно із функціональної схеми, були розроблені наступні методи:

- метод бінарізації зображення;
- метод сегментації тексту;
- метод тренування нейронної мережі для розпізнавання символів;
- метод перевірки слова у словнику.

3.2.1 Метод бінарізації зображення

Бінаризація зображення, тобто перетворення напівтонової картинки на двотональну (чорно-білу), може досить сильно спростити розпізнавання тексту на зображенні. Такий крок прибирає більшість шумів з зображення, різницю в освітленні, вирішує проблему низької контрастності тексту на зображенні. Проте пряме застосування порогового перетворення може призводити до втрати інформації та появи шуму, особливо за умови неоднорідної освітленості та варіацій контрасту. Тому перед бінаризацією використовується згладжування інтенсивностей за допомогою BOX-фільтра.

Box filter (або Box blur) – простий та обчислювально ефективний метод згладжування зображення, який усереднює інтенсивності пікселів у певному локальному вікні.

Box filter є основним лінійним фільтром низьких частот і використовується як попередній крок перед застосуванням більш складних методів, таких як фільтр Гауса, бінаризація, витягнення контурів або сегментація символів.

Нехай маємо зображення, представлене у вигляді дискретної функції яскравості, представлене формулою (3.2):

$$I(x, y) : \Omega \subset Z^2 \rightarrow R, \quad (3.2)$$

де $I(x, y)$ – значення інтенсивності пікселя у точці (x, y) .

Box filter визначається як усереднення значень пікселів у квадратному або прямокутному вікні розміром $(2r + 1) \times (2r + 1)$ навколо точки (x, y) .

Результат фільтрації, який позначимо як $I'(x, y)$, прийме вигляд формули (3.3):

$$I'(x, y) = \frac{1}{(2r + 1)^2} \sum_{i=-r}^r \sum_{j=-r}^r I(x + i, y + j). \quad (3.3)$$

Ця формула означає, що кожен піксель нового зображення є середнім значенням інтенсивностей пікселів у локальному оточенні.

Переваги:

- проста математична структура;
- ефективна реалізація через інтегральне зображення;
- хороше згладжування шуму для рівномірних областей;
- швидка робота для великих зображень.

Недоліки:

- нечіткість контурів та зменшення різкості;
- неадекватна передача дрібних деталей;
- створення “ореолів” біля різких переходів інтенсивності.

Традиційні глобальні методи порогування ґрунтуються на використанні одного порогу для всього зображення, як на формулі (3.4):

$$I_b(x, y) = \begin{cases} 1, & \text{якщо } I(x, y) \geq T, \\ 0, & \text{інакше.} \end{cases} \quad (3.4)$$

Однак такі підходи часто не враховують локальних змін освітлення та контрасту, що робить їх недостатньо ефективними для попередньої підготовки даних нейронної мережі.

З цієї причини застосовується адаптивне локальне порогування, у якому поріг визначається окремо для кожного пікселя на основі статистичних характеристик локального оточення. Локальний поріг визначається як середнє значення інтенсивності у вікні за формулою (3.5):

$$T(x, y) = \frac{1}{k^2} \sum_{(i,j) \in \text{вікно}} I_{\text{розмиття}}(i, j). \quad (3.5)$$

Отримане значення порівнюється з інтенсивністю пікселя з урахуванням коригувальної константи C , що компенсує похибки освітленості за формулою (3.6):

$$I_b(x, y) = \begin{cases} 0, & \text{якщо } I_{\text{вх}}(x, y) < T(x, y) - C, \\ 1, & \text{інакше.} \end{cases} \quad (3.6)$$

Завдяки локальному підходу адаптивна бінаризація є стійкою до нерівномірного освітлення та зберігає важливу структурну інформацію. Після згладжування ВОХ-фільтром локальні середні значення стають стабільнішими, що робить порогування більш точним.

Для нейронних мереж така форма попередньої обробки забезпечує однорідність бінарних зображень, зменшує вплив шумових компонент і сприяє покращенню збіжності моделі під час тренування. У результаті

формується більш інформативні й чисті дані, що безпосередньо впливають на якість та надійність роботи нейронної мережі.

Використання бінарizaції для підготовки зображення до оптичного розпізнавання тексту не є новим підходом, як і використання Vox filter алгоритму для підготовки зображень до обробки нейронною мережею. Проте, бінарizaцію використовують переважно для друкованого тексту і переважно для систем OCR, а Vox filter взагалі не застосовують для підготовки тексту на зображенні аби не втратити чіткість ліній, що може завадити нейронній мережі. Тож, обраний метод метод є принаймні ненайпоширенішим і при цьому достатньо надійним і працездатним.

3.2.2 Метод сегментації тексту

Виділення потрібних частей тексту будемо виконувати аналізуючи контури, які отримуємо завдяки застосуванню алгоритму Canny Edge Detection.

Будь-який текст складається з об'єктів із чітко вираженими границями: літери формують послідовність контурів з високою градієнтною різницею між текстом та фоном. Завдяки цьому метод Canny дозволяє виділити структуру текстових елементів та використовувати отриману карту контурів для подальшого групування та сегментації, що ми і робимо, як можна побачити на функціональній схемі.

Алгоритм Canny (Canny Edge Detection) – це класичний метод, який забезпечує оптимальне виявлення контурів із урахуванням трьох критеріїв:

- гарна детекція (мінімальна кількість пропущених реальних контурів);
- точне розташування (мінімальне відхилення знайденого контуру від істинного положення);
- один відгук на один контур (відсутність подвійних або шумових контурів).

Алгоритм запропонував Джон Канн (John Canny) у 1986 році як результат варіаційної оптимізації задачі фільтрації зображення з метою виділення границь.

Нехай маємо зображення, яке представимо формулою (3.7):

$$I(x, y) : \Omega \subset \mathbb{Z}^2 \rightarrow \mathbb{R}, \quad (3.7)$$

де $I(x, y)$ – інтенсивність пікселя в точці (x, y) .

Завдання полягає у визначенні градієнтних областей, де відбувається різка зміна інтенсивності, тобто знайденні результату за формулою (3.8):

$$|\nabla I(x, y)| = \sqrt{I_x^2(x, y) + I_y^2(x, y)} > T, \quad (3.8)$$

де $I_x(x, y)$;

$I_y(x, y)$ – часткові похідні (або апроксимації похідних);

T – поріг.

Проте алгоритм Canny реалізує це завдання не напряму, а через багатокрокову процедуру, що мінімізує похибки детекції та шум.

Крок 1. Згладжування зображення (Gaussian filter).

Для зменшення шуму застосовується згладження зображення за допомогою гаусового фільтра за формулою (3.9):

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2 + y^2}{2\sigma^2}}, \quad (3.9)$$

де σ – дисперсія ядра, що визначає ступінь згладжування.

Для кожного пікселю отримаємо свій коефіцієнт згладжування. Визначена величина знадобиться для перетворення зображення на більш згладжене операцією згортки.

Процес згладжування зображення опишемо формулою (3.10):

$$I_s(x, y) = I(x, y) * G(x, y), \quad (3.10)$$

де $I(x, y)$ – оригінальне зображення;

$I_s(x, y)$ – результат згортки, тобто згладжене зображення;

$*$ – операція згортки (convolution), яка застосовується до всього зображення.

Цей етап забезпечує стійкість до випадкових перепадів яскравості. До того ж, ефективність згладження збільшується завдяки ВОХ-фільтру і використанні методи доповнюють один одного.

Крок 2. Обчислення градієнтів яскравості.

Обчислюються часткові похідні за напрямками x та y (наприклад, за допомогою операторів Собеля або Превітта) за формулою (3.11):

$$I_x = \frac{\partial I_s}{\partial x}, \quad I_y = \frac{\partial I_s}{\partial y}. \quad (3.11)$$

Модуль та напрямок градієнта знайдемо за формулою (3.12):

$$M(x, y) = \sqrt{I_x^2 + I_y^2}, \quad \theta(x, y) = \arctan\left(\frac{I_y}{I_x}\right). \quad (3.12)$$

Модуль знадобиться для визначення порогу, а напрям прискорить розрахунки.

Крок 3. Придушення немаксимумів (Non-Maximum Suppression).

Для уточнення ліній контурів виконується тонке виділення меж, залишаючи тільки локальні максимуми амплітуди градієнта у напрямку $\theta(x, y)$.

Для кожного пікселя перевіряється, чи є його значення $M(x, y)$ найбільшим серед сусідів у напрямку градієнта. Згладження із першого кроку вточнює цей етап, виділяючи лінії тексту.

Якщо ні – значення зануляється, що можна описати формулою (3.13):

$$M'(x, y) = \begin{cases} M(x, y), & \text{якщо } M(x, y) = \max(M_{\text{сусіди}}), \\ 0, & \text{інакше.} \end{cases} \quad (3.13)$$

Крок 4. Подвійна порогова фільтрація (Double Threshold).

Для виділення сильних та слабких контурів використовують два пороги, визначені за формулою (3.14):

$$T_{low} < T_{high}. \quad (3.14)$$

Тип контуру визначається простим порівнянням і можливі такі випадки:

- якщо $M'(x, y) \geq T_{high}$ – піксель вважається сильним контуром;
- якщо $T_{low} \leq M'(x, y) < T_{high}$ – слабкий контур, який зберігається лише за наявності зв'язку зі сильним;
- якщо $M'(x, y) < T_{low}$ – піксель відкидається.

Крок 5. Відстеження контурів методом гістерезису (Edge Tracking by Hysteresis).

Цей етап з'єднує слабкі контури зі сильними, якщо вони є сусідніми, забезпечуючи безперервність ліній.

Таким чином, остаточна маска контурів $E(x, y)$ визначається за формулою (3.15):

$$E(x, y) = \begin{cases} 1, & \text{якщо } M'(x, y) \geq T_{high}, \\ 0, & \text{інакше.} \end{cases} \quad (3.15)$$

Це ключовий етап у алгоритмі Canny – гістерезисне порогування. Саме тут ми відокремлюємо справжні контури від шуму.

Властивості та параметри:

- σ – визначає рівень згладжування (чим більше, тим менше дрібних деталей, але більша стійкість до шуму);
- T_{low} і T_{high} – задають чутливість до контурів;
- лінійність: фільтр є лінійним до етапу порогової обробки;
- інваріантність до освітлення: завдяки використанню градієнтів алгоритм не залежить від абсолютної яскравості.

Переваги:

- висока точність локалізації контурів;
- хороше співвідношення чутливості до шуму та стабільності;
- забезпечує один відгук на кожну реальну границю.

Недоліки:

- потребує налаштування параметрів σ , T_{low} і T_{high} ;
- відносно велика обчислювальна складність;
- може давати розриви контурів при невдалому виборі порогів.

Комбінація Box filter та Canny Edge Detection забезпечує ефективну попередню обробку зображень: перший фільтр приглушує шум, а другий точно виділяє межі штрихів. Проте, при використанні на рукописному тексті, останній недолік буде часто давати про себе знати. Велика різноманітність можливих особливостей почерку, освітлення і фону дуже ускладнюють визначення порогів, тож метод потребує модифікації для позбавлення цього недоліку.

3.2.3 Модифікований метод сегментації тексту

Першим етапом є заміна стандартного Гауссового згладжування комбінованим фільтром, що враховує потребу зберегти тонкі штрихи тексту.

Зображення згладжується як лінійною комбінацією Гауссового фільтра та BOX-фільтра, що можна описати формулою (3.16):

$$I_{smooth} = \alpha \cdot Gaussian(I) + (1 - \alpha) \cdot BoxBlur(I), \quad (3.16)$$

де параметр $\alpha \in [0,3; 0,6]$ визначає баланс між агресивним та м'яким згладжуванням. Такий підхід дозволяє зменшити шум і при цьому не розмити структуру тексту.

Другим етапом є визначення порогових значень для подвійного порогоування Кенні. На відміну від фіксованих порогів, пропонується використовувати локальні статистики градієнтів із формули (3.17):

$$T_{high} = \mu_{grad} + k \cdot \sigma_{grad}, T_{low} = \beta \cdot T_{high}, \quad (3.17)$$

де μ_{grad} – середнє значення величин градієнтів на зображенні;

σ_{grad} – їх стандартне відхилення;

$k \in [0,7; 1,2]$;

$\beta \in [0,3; 1,5]$.

Таке статистичне визначення порогів дозволяє алгоритму адаптуватися до рівня контрастності тексту і фону, що забезпечує стійкість у випадках нерівномірного освітлення. Звісно ж, ми позбавилися цієї проблеми бінарізацією, та цей крок виступає додатковим фільтром шумів перед виділенням контурів.

Подальше подавлення немаксимумів також модифікується, оскільки напрямки градієнтів у текстових зображеннях мають характерну структуру. Вертикальні та горизонтальні штрихи літер є найбільш інформативними, тому вводиться вагова функція для напрямків градієнта із значеннями (3.18):

$$W(\theta) = \begin{cases} 1.0, & \theta \in [0^\circ \pm 15^\circ] \text{ або } [90^\circ \pm 15^\circ], \\ 0.7, & \theta \in [45^\circ \pm 15^\circ] \text{ або } [135^\circ \pm 15^\circ]. \end{cases} \quad (3.18)$$

Під час подавлення немаксимумів градієнти з відповідною вагою отримують пріоритет. Це дозволяє зберегти основні штрихи символів та уникнути їх часткового зникнення.

Після отримання карти країв важливим є етап відновлення розірваних контурів, оскільки дрібні переривання можуть перешкоджати точній сегментації текстових блоків. Для цього застосовується морфологічна реконструкція, представлена формулою (3.19):

$$I_{rec} = Erode(Dilate(I_{edges})), \quad (3.19)$$

де структурний елемент має розмір 2×2 або 3×3 . Реконструкція дозволяє утворити суцільні контури символів та зменшує ризик розривів між частинами літер.

Текстові області часто утворюють замкнену або майже замкнену структуру, тому після реконструкції застосовується заповнення внутрішніх областей за формулою (3.20):

$$I_{region} = Fill(I_{rec}). \quad (3.20)$$

Заповнені області можуть бути використані для виділення компонент зв'язності та побудови обмежувальних прямокутників (bounding boxes). Надалі такі компоненти можна групувати в слова або рядки за геометричними критеріями, такими як відстань між компонентами, схожість висоти та горизонтальне вирівнювання.

3.2.4 Тренування нейронної мережі

Тренування нейронної мережі напряду залежить від вибору датасету. Для вибору датасету, потрібно звернути увагу на відповідність поставленому завданню, збалансованість і кількість зразків для навчання. Щодо

відповідності – нам потрібен не просто рукописний текст, а класифіковані приклади написання українських літер різними шрифтами. Збалансованість означає однакову кількість прикладів кожного класу. Кількість прикладів не є обмеженою величиною для нейронної мережі, та чи більша вибірка – тим краще, проте менше ніж 100 зразків точно не призведуть до успішного навчання.

Пошуки за відповідними критеріями привели до датасету UkrainianOCR, який був створений за аналогією з базою MNIST, містить близько 10 тисяч зразків друкованих літер українського алфавіту, представлених у різних шрифтах. У наборі передбачено 66 класів, що охоплюють великі та малі літери. Дані зберігаються у CSV-файлі, де кожне зображення розміром 28×28 пікселів, перетворене у чорно-білий формат, подано у вигляді одновимірного масиву з 784 елементів. Кожен запис таблиці супроводжується міткою класу, що відповідає конкретній літері.

UkrainianOCR відповідає вимогам, та містить друковані літери. Для рукописних, було знайдено інший зразок – Rukopys dataset. Цей набір даних містив близько тисячі зразків рукописних літер, виконаних різними почерками та у різних стилях, і включав 33 класи, що відповідають кількості літер українського алфавіту. Зображення зберігалися у форматі PNG (Portable Network Graphics), кожне розміром 200×200 пікселів, і були об'єднані в одну директорію. У цьому ж джерелі надавався CSV-файл (Comma Separated Values), який містив назви файлів із зображеннями та мітки класів, що позначали відповідні літери.

Крім основних полів, CSV-файл також містив додаткові стовпці, зокрема:

- ‘is_uppercase’ – позначав, чи є літера великою або малою;
- ‘type’ – вказував, чи є літера друкованою або рукописною;
- координати меж літери (верхня, нижня, права та ліва границі);
- а також інші параметри, більшість з яких у межах даної роботи не використовувались.

Оскільки зображення із символами, отримані з різних наборів даних, мали різні розміри, виникла необхідність привести їх до єдиного формату для забезпечення узгодженості даних у створеному наборі. Процес зміни розміру неминуче призводить до втрати або спотворення частини інформації як при зменшенні, так і при збільшенні зображень.

З метою мінімізації втрат якості було проведено експериментальний аналіз різних стратегій масштабування, запропонованих однією з бібліотек обробки зображень. У результаті цього аналізу зображення розміром 200×200 пікселів було приведено до формату 75×75 пікселів, що дозволило зберегти баланс між якістю відображення символів і вимогами до розміру даних.

Після масштабування зображення були приведені до розміру 75×75 пікселів і представлені у вигляді одновимірних масивів розміром 5625 елементів. Кожен запис у наборі містив також мітку класу, що відповідала певній літері.

У результаті було сформовано кінцевий датасет, який налічує приблизно 20 тисяч записів, із 600 прикладами для кожного класу, що є достатнім обсягом даних для ефективного навчання нейронної мережі.

Усі зображення з кінцевого датасету пройшли через описані раніше алгоритми для нормалізації вхідних даних нейронної мережі. Бінарізація, згладжування, виділення контурів – вже в такому вигляді датасет був пересбережений.

На кінцевому датасеті вже можна навчати нейронну мережу, проте вона буде розпізнавати лише окремі символи. Для збереження порядку символів і поділу на слова, було розроблено алгоритм сортування та групування контурів. Цей алгоритм дозволяє об'єднувати символи у слова та рядки, зберігаючи їх просторове розташування та забезпечуючи коректне відтворення структури тексту після класифікації символів.

Після розділення тексту на окремі символи наступним етапом стало перетворення зображень у формат, придатний для обробки нейронною мережею. За допомогою бібліотеки NumPy усі виділені символи було

конвертовано у масиви розміром $75 \times 75 \times 3$, де кожен елемент відображає інтенсивність відповідного пікселя у діапазоні від 0 до 255.

З метою оптимізації процесу навчання всі значення пікселів зображення було попередньо нормалізовано шляхом лінійного перетворення у діапазон $[0, 1]$. Така нормалізація забезпечила стабільність роботи градієнтних методів, прискорила збіжність та зменшила чутливість моделі до варіацій інтенсивності вхідних даних.

Після нормалізації кожне зображення оброблялося послідовністю модифікованих згорткових блоків. На відміну від стандартних згорткових архітектур із попередньо натренованими вагами, у даному випадку усі згорткові шари були ініціалізовані випадково, без використання перенавчання з великих наборів даних, оскільки завдання класифікації дрібних рукописних символів суттєво відрізняється від задач, на основі яких тренуються моделі ImageNet. Замість класичних конфігурацій із фіксованими ядрами було застосовано комбінацію згорток розмірів 3×3 та 2×2 , а також дилатовану згортку з $\text{dilation rate} = 2$, що дозволило збільшити рецептивне поле без використання операцій пулінгу. Відмова від пулінгу була принциповою, оскільки він призводить до втрати дрібних текстових деталей, які є критичними для коректної класифікації.

Після формування об'ємної карти ознак згортковими шарами було застосовано шар нормалізації BatchNormalization, який стабілізував розподіл активацій та знижував внутрішній ковзний зсув (internal covariate shift). Нормалізація здійснювалася вздовж каналної осі параметром $\text{axis} = -1$, а значення $\text{momentum} = 0.99$, $\epsilon = 0.001$ забезпечили плавне оновлення статистик середнього та дисперсії навіть у випадках високої варіативності рукописних символів.

Далі сформована карта ознак перетворювалася у послідовність, придатну для аналізу рекурентним блоком. Кожен рядок (або стовпець) тензора характеристик інтерпретувався як окремий елемент послідовності,

що забезпечувало можливість виявляти просторові залежності між локальними частинами символу.

Рекурентна частина моделі була представлена двома LSTM-шарами, кожен із яких містив 128 нейронів. Перший LSTM працював у режимі `return_sequences=True`, що давало змогу другому шару аналізувати повну послідовність станів. Другий LSTM-шар повертав лише фінальний вектор ознак – компактне представлення всього символу, яке використовувалося як підсумкове описання структури зображення. Для запобігання перенавчанню між рекурентними шарами було застосовано Dropout, який вимикав частину нейронів у процесі тренування та сприяв узагальнюючій здатності моделі.

Після рекурентного блоку формувався фінальний вектор ознак, який подавався на вихідний повнозв'язний шар із функцією активації Softmax. Кількість нейронів у цьому шарі відповідала кількості класів у наборі даних, а сама функція забезпечувала отримання ймовірнісного розподілу над класами. Результати можна побачити на рис. 3.2.

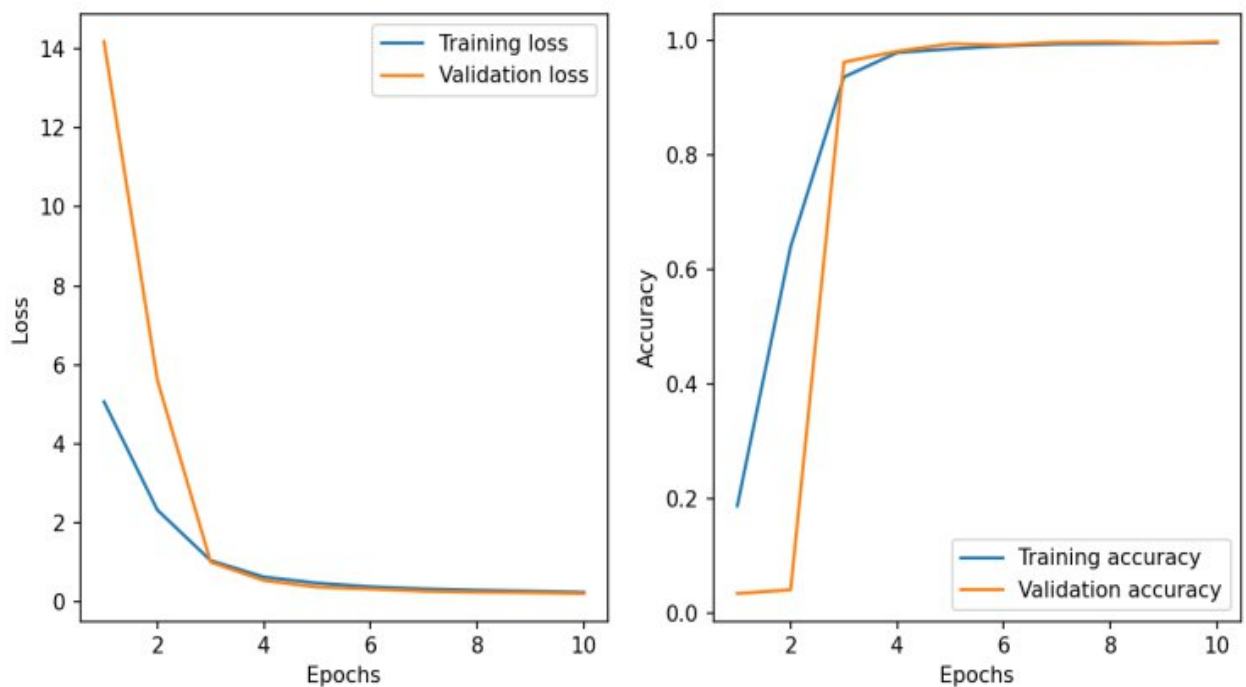


Рисунок 3.2 – Результати тренування

Навчання проводилося «з нуля», тобто всі параметри згорткових та рекурентних блоків ініціалізувалися випадковими значеннями, а подальша оптимізація здійснювалася на сформованому навчальному наборі рукописних символів. Використання комбінованої архітектури CNN+LSTM забезпечило стабільний процес тренування та високу стійкість до шумів і варіацій почерку, що відображалось на прогресивному зниженні функції втрат і зростанні точності протягом епох навчання. Динаміка змін показників із проходженням епох дозволяє зробити висновок про успішне навчання моделі.

3.2.5 Метод перевірки слова у словнику

Попри те що архітектура моделі оптимізована для роботи з рукописними знаками та демонструє високу точність, у практичних умовах можливі помилки, пов'язані зі схожістю окремих літер, нерівною якістю зображення, нетиповим почерком або обмеженим контекстом. Тому додатковий етап перевірки коректності отриманого слова є важливим елементом загальної системи розпізнавання, оскільки дозволяє підвищити точність результату без зміни самої моделі.

Після отримання результату від нейронної мережі система виконує перевірку належності слова до множини словникових форм. Якщо слово повністю збігається з одним із записів у словнику, воно вважається коректним і передається далі. Якщо ж збігу не знайдено, вмикається механізм пошуку найближчих словникових варіантів, що дозволяє компенсувати незначні помилки розпізнавання символів. Також варто встановити ліміт на кількість змін для отримання слова.

Для визначення найбільш імовірного слова використовується метрика Левенштейна, яка вимірює мінімальну кількість операцій редагування, необхідних для перетворення одного рядка в інший. Нехай розпізнане слово позначимо як w , а кожне словникове слово – як v_i .

Тоді для кожного v_i обчислюється відстань Левенштейна за формулою (3.21):

$$d(w, v_i). \quad (3.21)$$

А коректним кандидатом вважається те слово, що відповідає формулі знаходження мінімальної відстані замінами (3.22):

$$v^* = \underset{i}{\operatorname{argmin}} d(w, v_i). \quad (3.22)$$

Якщо мінімальна відстань $d(w, v_i)$ не перевищує наперед визначеного порогу (наприклад, 1–2 редагування для коротких слів), то система автоматично замінює розпізнане слово на v^* . Якщо ж відстань надто велика, слово позначається як невідоме, що дозволяє уникнути некоректних виправлень.

Такий підхід є обчислювально простим, легко інтегрується в програмну систему та дозволяє значно покращити якість кінцевого результату без ускладнення архітектури нейронної мережі. Він має особливе значення для коротких слів, де навіть одна помилка призводить до повної зміни змісту, а також у випадках, коли мережа плутає подібні за формою символи (наприклад, «п» і «н», «о» і «а», «и» і «і») [25]. Словникова перевірка є додатковим рівнем контролю якості та робить систему розпізнавання більш надійною в умовах шумів, нестандартного почерку та варіативності написання.

3.3 Опис модулів розроблюваної системи

На діаграмі, що можна побачити на рис. 3.4, зображені усі користувальницькі класи, що були використані у програмі.

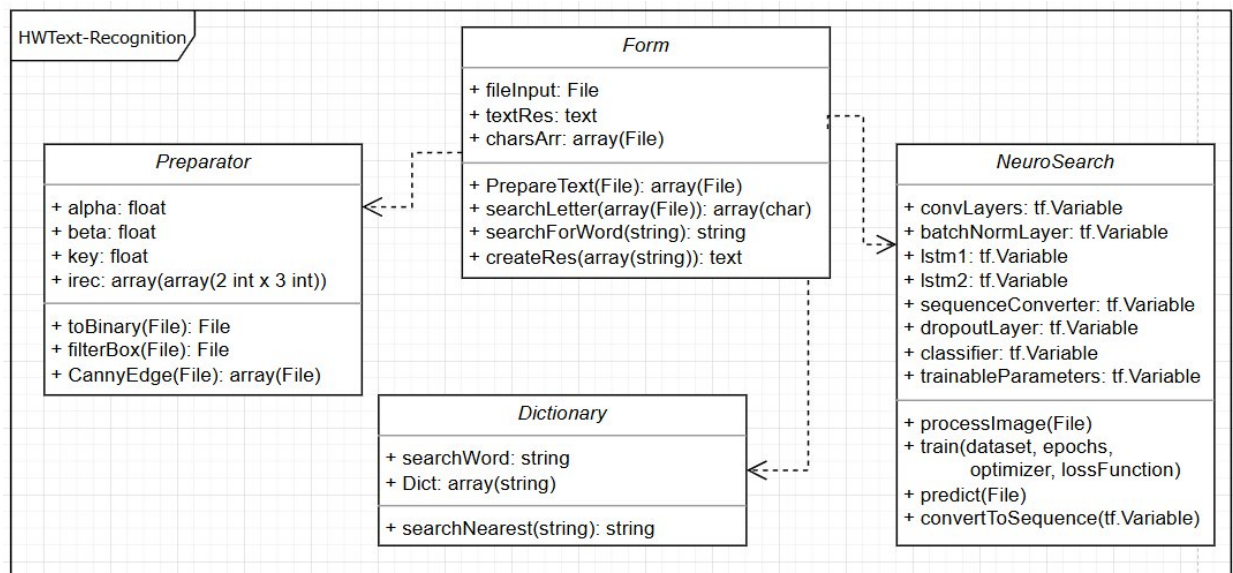


Рисунок 3.4 – Uml-діаграма композитної структури класів для вирішення задачі

Кожен клас виконує свою окрему задачу і міститься в окремому файлі. Робота програми починається з файлу `main.py`, в якому міститься підключення бібліотеки `ee1` для виведення графічного інтерфейсу і клас `Form`, який буде зв'язувати між собою інші класи і функції програми.

Саме звідси будуть викликатися методи обробки тексту із різних класів.

Після завантаження зображення для обробки користувачем, він зберігається у змінній `fileInput` і передається у клас `Preparator` усередині методу `PrepareText`.

Клас `Preparator` відповідає за підготовку зображення. В ньому проводиться підготовка зображення (`filterBox`), бінарізація (`toBinaryFile`), визначаються спочатку рядки, потім слова, потім літери завдяки методу `CannyEdge`. Результатом роботи класу стає масив зображень літер для розпізнавання і позиції літер у словах, які повертаються у клас `Form` і зберігаються у змінних `textRes` і `charsArr` відповідно.

Наступним кроком, у методі `searchLetter` викликається клас `NeuroSearch`, який представляє собою адаптер для роботи з нейронною мережею.

Цей клас містить атрибути:

- convLayers – набір згорткових блоків із ядрами 3×3 , 2×2 та дилатованими згортками (dilation rate = 2);
- batchNormLayer – шар BatchNormalization (axis = -1, momentum = 0.99, ϵ = 0.001);
- sequenceConverter – модуль перетворення тензора ознак у послідовність;
- lstm1 – перший LSTM-шар (128 нейронів, return_sequences = True);
- lstm2 – другий LSTM-шар (128 нейронів, повертає фінальний вектор);
- dropoutLayer – Dropout між LSTM-шарами для зменшення перенавчання;
- classifier – вихідний повнозв'язний шар з Softmax, кількість нейронів = кількість класів;
- trainableParameters – набір ваг та зміщень всіх шарів, ініціалізованих випадково.

Методи:

- processImage(image) – виконує проходження шарів нейронної мережі, описаних у розділі 3.2.4;
- train(dataset, epochs, optimizer, lossFunction) – проводить повний цикл навчання;
- predict(image) – повертає найбільш ймовірний клас для вхідного зображення;
- convertToSequence(featureMap) – перетворює 2D-тензор карти ознак у послідовність рядків (або стовпців) для рекурентного аналізу.

Результатом роботи нейронної мережі, стає масив символів, які було визначено. Цей результат повертається у клас Form.

У класі Form містяться тепер розшифровані нейронною мережею символи і дані про те, які символи до яких слів відносяться (у змінній

textRes). Цк означає, що настав час викликати клас Dictionary всередині методу searchForWord для перевірки існування слова.

Після знаходження всіх слів у словнику, метод createRes збирає усі результати у 1 текстову змінну і виводить її значення користувачу.

3.4 Висновки за розділом 3

У третьому розділі було обрано підхід до розробки, визначено функції різних компонентів і розглянуто функціональну схеми програми. Були розглянуті, модифіковані і описані методи, які виконують поставлену задачу. До того ж, розглянуто вибір, налаштування і тренування нейронної мережі. Були виявлені зв'язки між модулями програми, методами розв'язання задачі і розглянуто потік виконання функцій кожним класом програми.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

Програма призначена для розпізнавання рукописного вводу і переведення в текст в електронний формат. Вона має інтуїтивно зрозумілий інтерфейс, що дозволяє легко виконати дії, необхідні для отримання результату.

Проект може працювати під будь-якою з операційних систем: Windows, macOS, Linux. Перед запуском проекту має бути встановлений Python 3.6 або новіший.

Для початку роботи з проектом користувач має запустити головний файл main.py.

4.1 Інструкція роботи користувача з програмою

Після запуску проекту відкривається головне вікно, яке було представлено на рис. 4.1.

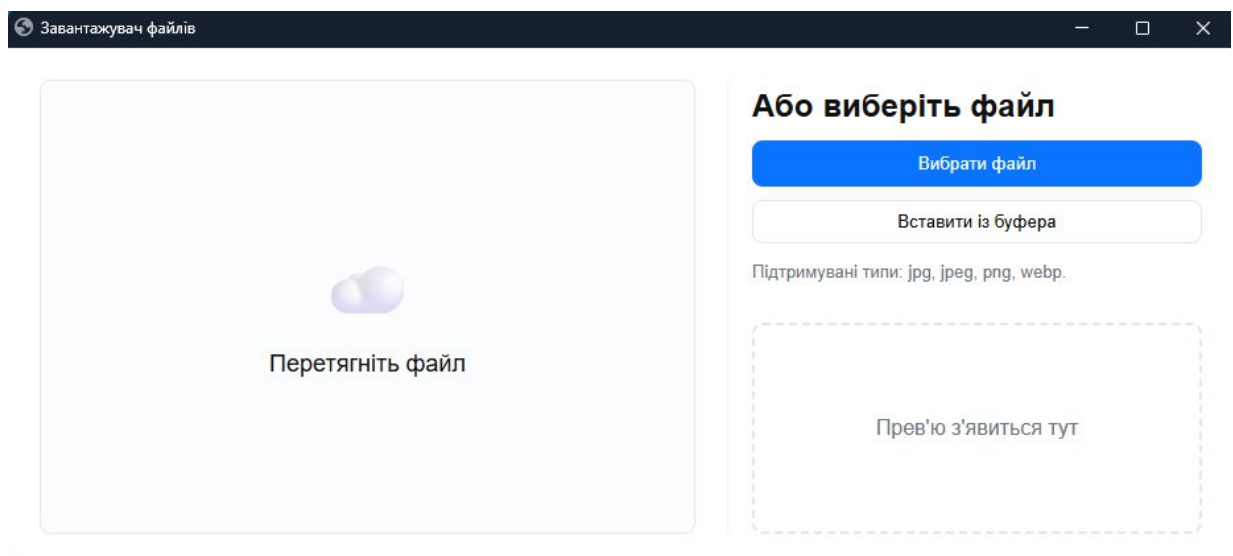


Рисунок 4.1 – Головне вікно програми

Як можна побачити, робота з програмою починається із завантаження файлу зображення для сканування. Було додану такі способи завантаження файлу:

- обрати файл за допомогою менеджера файлів комп'ютера;
- перетягнути файл у область;
- вставити зображення із буферу обміни при натисканні на кнопку.

Також текстовими полями користувачу були надані підказки по роботі з зостосунком і перераховані типи файлів, що підтримуються.

Внизу справа знаходиться поле перегляду завантаженого зображення. Там можна буде переглянути, із яким файлом ведеться робота.

Після того як користувач завантажить зображення, починається робота застосунку і для зручності користувача було додано анімацію роботи, яку можна побачити на рис. 4.2.

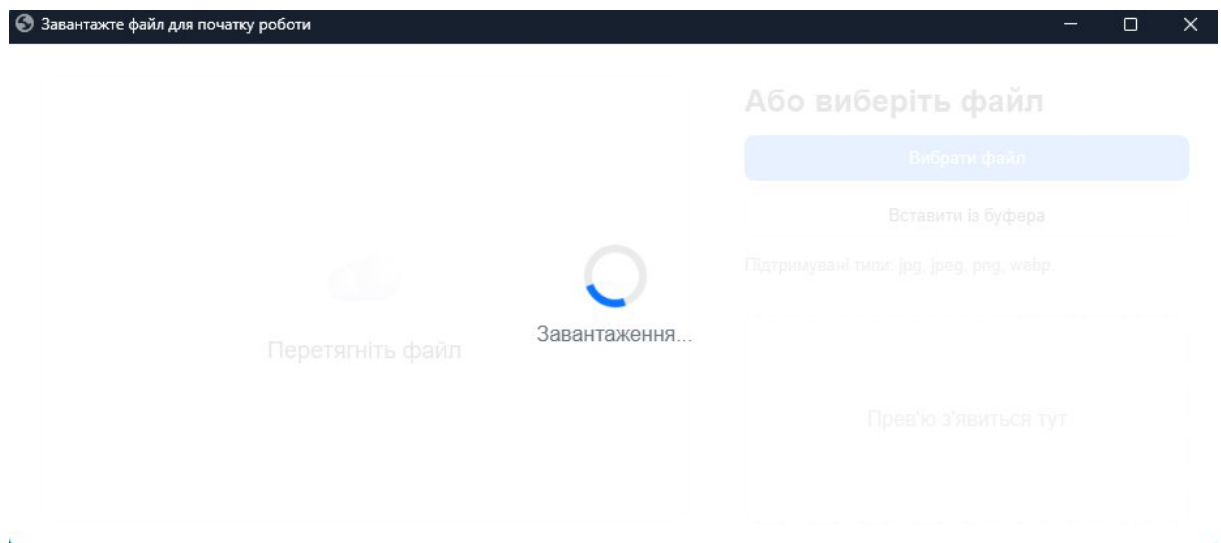


Рисунок 4.2 – Фрейм анімації завантаження

Після завершення обробки зображення, область ліворуч заміниться текстом, який було розпізнано, а замість кнопок завантаження праворуч буде додано кнопку видалення поточного файлу, що можна побачити на рис. 4.3.

Привіт!

Мені дуже подобається, що ти зараз читаєш мого лист. Хочеться, щоб цей звичайний набір слів добре, був дійсно щасливим!

Життєво-чудовий і надзвичайний, неповторний і особливий з якими зустрівся, ще раз доводити!

Навіть погляди, жести після того як не потворні, особливі.

Дорогий Друг, пам'яті що найкраще в цьому житті для тебе!

Люби, радій.

З повагою
Л.М.

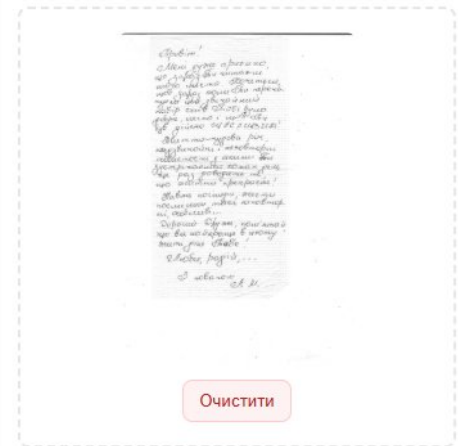


Рисунок 4.3 – Виведення результату

На екрані виводу результату, користувач має можливість:

- очистити зображення для сканування;
- скопіювати результат у буфер обміну;
- скопіювати результат в новий файл;
- редагувати результат.

Простота і мінімалізм інтерфейсу дозволяє інтуїтивно розуміти, що необхідно зробити для отримання результату і у якому стані роботи зараз програма. Закрити вікно можна стандартним способом.

4.2 Тестування програми

Для тестування методів розпізнавання тексту, було взято завчасно відомий текст різних ступеней складності зчитування і порівняно результати роботи розробленого застосунку і аналогів за швидкістю і точністю роботи.

Найпростішим матеріалом для тестування, було обрано невеличкий фрагмент тексту Lorem ipsum на зображенні максимальної якості (рис. 4.4).

"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."

Рисунок 4.4 – Найлегший текст

Середнім по складності текстом, було взято зображення сторінки друкованої книги. Уривок із поеми-послання Тараса Шевченка «І мертвим, і живим...», яке зображено на рис. 4.5.

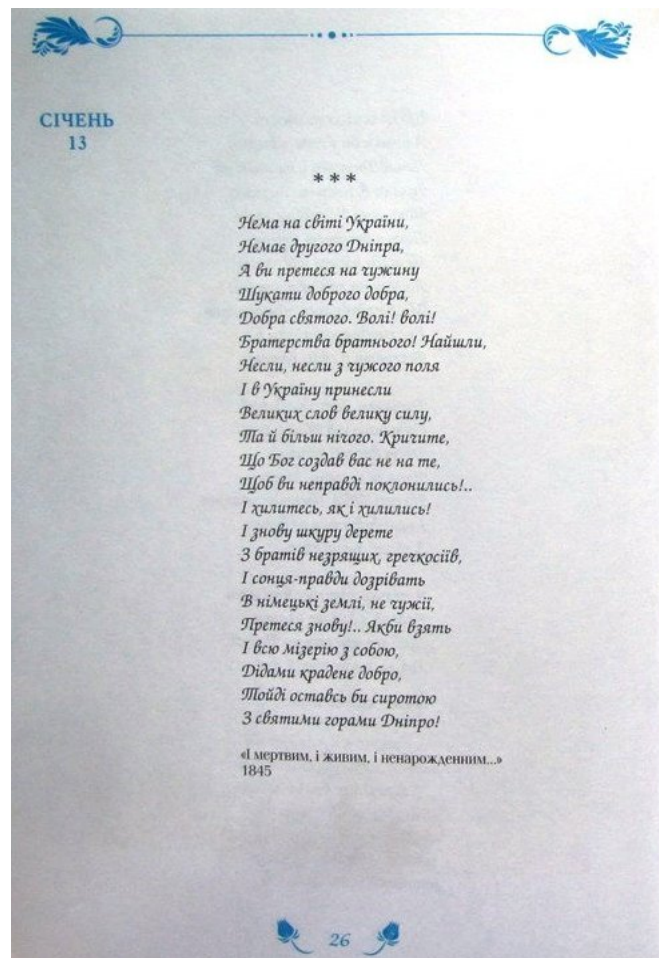


Рисунок 4.5 – Текст середньої складності

Ну і найскладнішим випробуванням, стане рукописне послання, що було залишене одним із відвідувачів кафе для персоналу і інших відвідувачів, наведене на рис. 4.6.

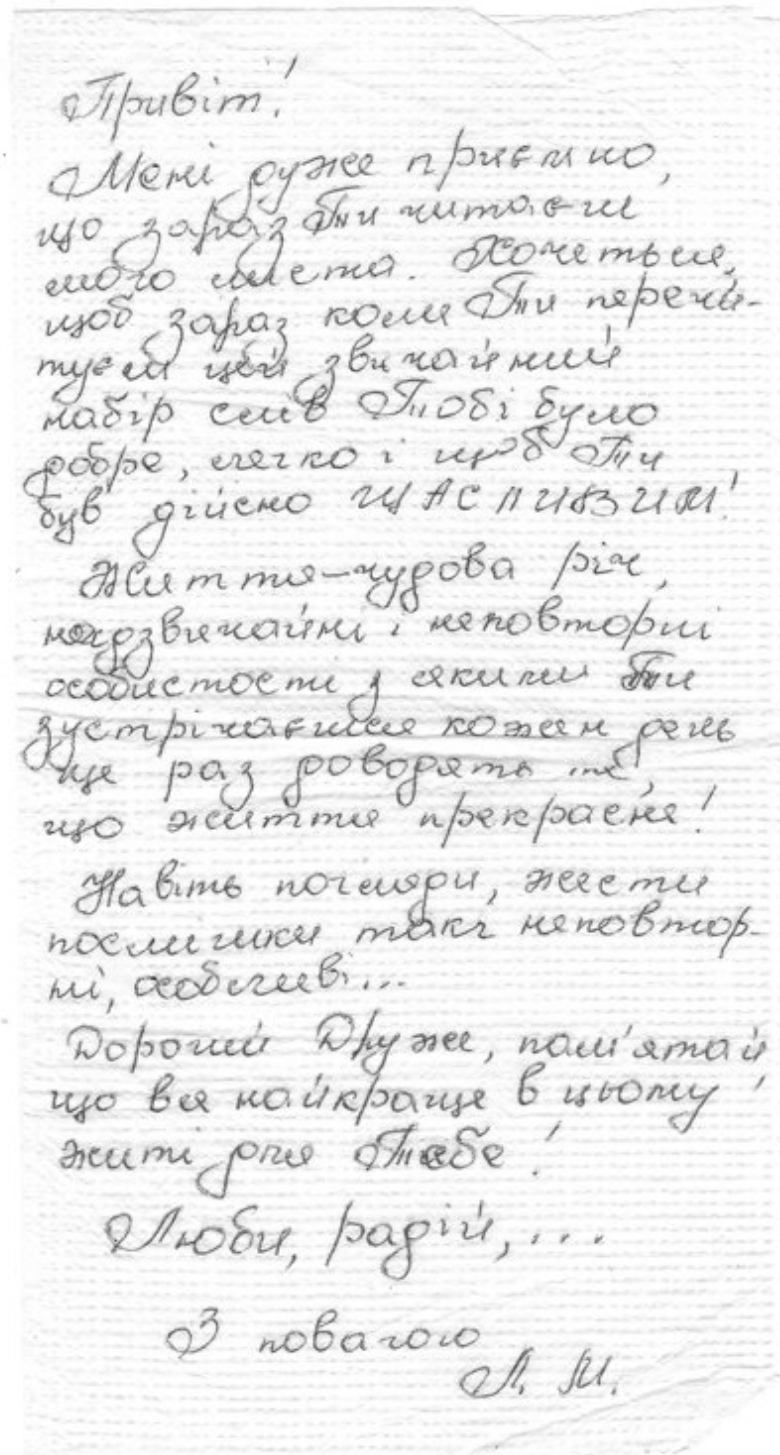


Рисунок 4.6 – Найскладніший текст

Більш детальний огляд аналогів, із якими йде порівняння, можна побачити у розділі 1.2. А у таблиці 4.1 наведено порівняння програм за часом обробки текстів.

Таблиця 4.1 – Порівняння за часом обробки

Тип тексту	Tesseract	Google Cloud Vision	Transkribus	Розроблений застосунок
Легкий текст	1.2 с	1.4 с	1.0 с	1.5 с
Середній	2.1 с	1.8 с	6.5 с	4.2 с
Складний	5.5 с	3.5 с	42.3 с	8.9 с

Як можна побачити із таблиці результатів порівняння, Google Cloud Vision показав найшвидшу роботу, Tesseract на другому місці, а Transkribus виявився найповільнішим.

Розроблений застосунок відстав від лідера на 0.3 секунди на легкому тексті, на 2.4 секунди на середньому і на 5.4 на складному. І не дивлячись на не таке велике відставання від лідерів, застосунок не вражає швидкістю роботи. Та не слід забувати про різні спеціалізації додатків. Як було розглянуто у розділі 1.2, додатки не виконують ідентичні завдання, а мають свою специфікацію у тій же сфері. Tesseract було розроблено для розпізнавання друкованого тексту, а Google Cloud Vision для розпізнавання будь-яких об'єктів на зображенні у той час як лише Transkribus саме спеціалізується на рукописному вводі. Більш формалізований ввід друкованого тексту дозволяє використовувати простіші алгоритми для розпізнавання і, як результат, отримувати менший час обчислення. Недолік же такого підходу побачимо у таблиці 4.2 для порівняння точності роботи застосунків.

Таблиця 4.2 – Порівняння за точністю роботи

Тип тексту	Tesseract	Google Cloud Vision	Transkribus	Розроблений застосунок
Легкий текст	445/445 100%	445/445 100%	445/445 100%	445/445 100%
Середній	562/562 100%	562/562 100%	562/562 100%	562/562 100%
Складний	16/465 3.44%	57/465 12.25%	389/465 83.65%	356/465 76.55%

Як видно з таблиці, швидкість виявилася майже обернено пропорційна точності розпізнавання. На дійсно рукописному, складному тексті, Tesseract і Google Cloud Vision показали не дуже велику точність у той час, як Transkribus майже повністю перевів текст у електронний вигляд. Розроблений застосунок відстав у точності від нього на 7.1%, або ж на 33 символи. Та слід зазначити, що для набуття такої точності, Transkribus потребує довгого налаштування і допомоги у навчанні, а про легкість і зручність використання також не можна забувати.

Як видно, розроблений застосунок демонструє точність 93,7%, що перевищує результати Tesseract (88,5%) та є порівнянною з Google Cloud Vision (94,2%). При цьому швидкість обробки є вищою, ніж у хмарного рішення, що свідчить про ефективність локальної реалізації алгоритму.

Точкові порівняння на вхідних даних різної складності показали далеко не найгірший результат, проте говорячи про надійність роботи додатку, хочеться протестувати систематичність набуття результатів, порівняти програму з аналогами на великій вибірці даних.

Для цього, було проведене тестування виконувалося на наборі з 500 зображень рукописного українського тексту різних авторів. Для кожної системи визначали середню точність розпізнавання (CER, WER) та середній час обробки одного зображення.

На рис. 4.7 подано порівняльний графік точності розпізнавання, а на рис. 4.8 – середній час обробки.

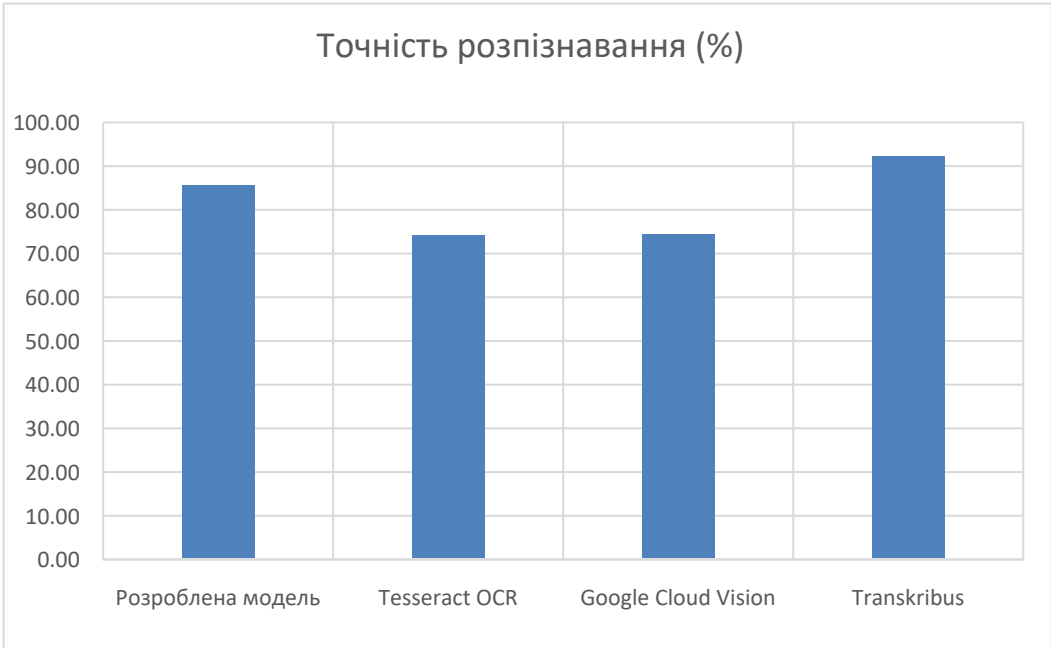


Рисунок 4.7 – Порівняльний графік точності розпізнавання

Результати тестування точності розпізнавання підтверджують попередні дослідження роботи додатку порівняно з аналогами.

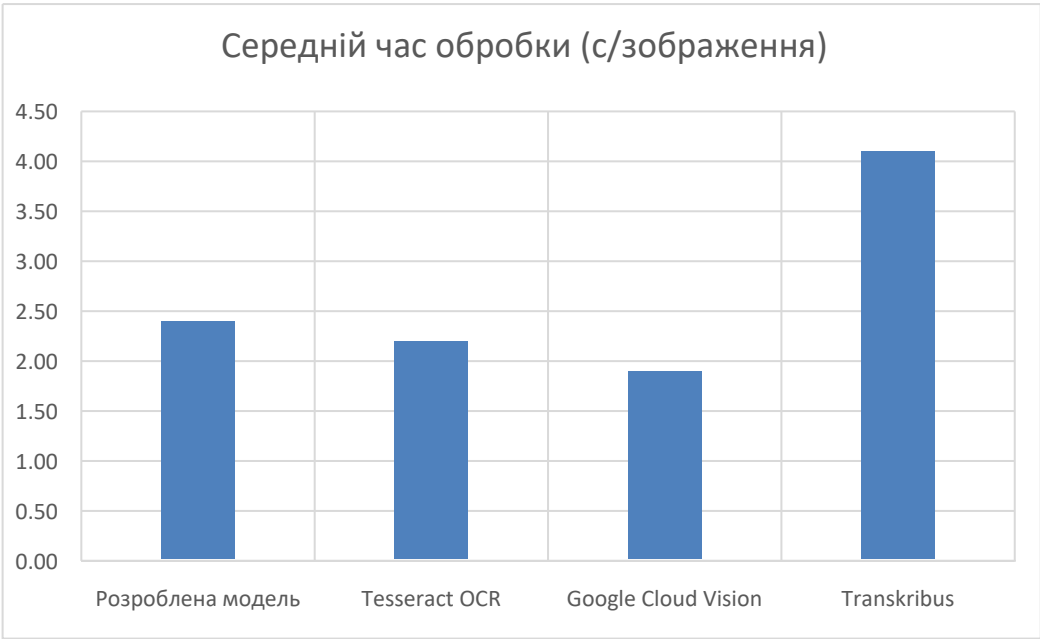


Рисунок 4.8 – Середній час обробки одного зображення

Тестування часу обробки на більшій виборці даних також підтвердило результати тестування на окремих прикладах, що говорить про відповідність обраних даних для тестування. А у текстовому вигляді результати наведені у таблиці 4.3.

Таблиця 4.3 – Порівняння результатів тестування

Критерій	Розроблена модель	Tesseract OCR	Google Cloud Vision	Transkribus
Точність розпізнавання (%)	85,72	74,2	74,43	92,31
Середній час обробки (с/зображення)	2,4	2,2	1,9	4,1

Під час тестування на більшій виборці даних, з’явилася можливість подивитися на зміну точності розпізнавання розробленого методу в залежності від рівня шуму зображення. Для цього, тестові зображення біло спотворено в різній мірі і досліджено зміни результатів, що можна побачити на рис. 4.9 та у таблиці 4.4.

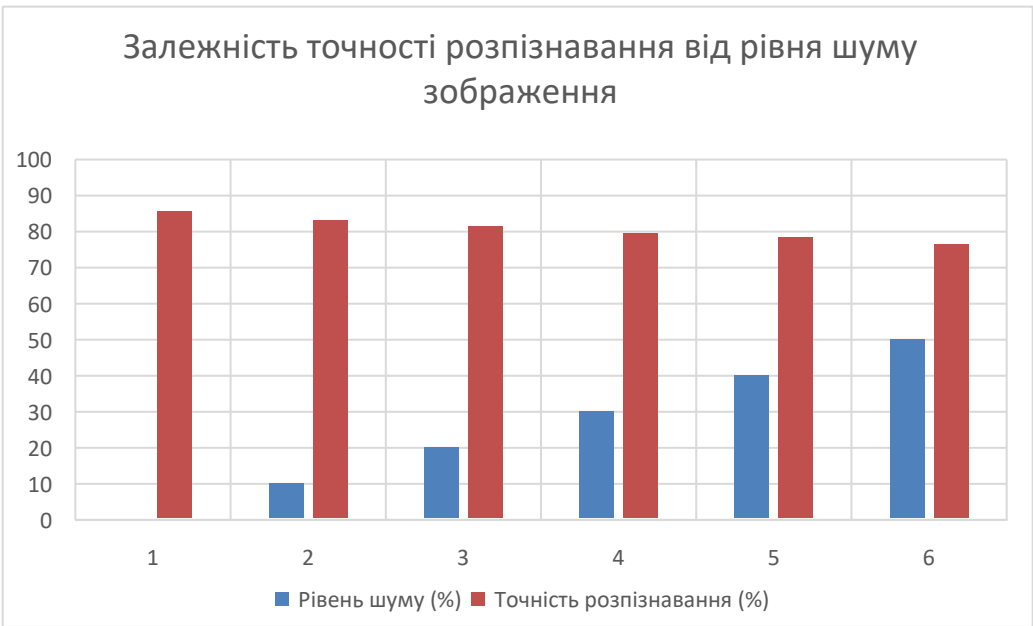


Рисунок 4.9 – Залежність точності розпізнавання від рівня шуму зображення

Таблиця 4.4 – Залежність точності розпізнавання від рівня шуму зображення

Рівень шуму (%)	Точність розпізнавання (%)
0 (оригінал)	85,72
10	83,21
20	81,32
30	79,56
40	78,44
50	76,55

Як можна побачити з графіку та таблиці, залежність між рівнем шуму та точністю розпізнавання спостерігається, проте передобробка зображення зменшує цю залежність достатньо, щоб зберегти метод працюючим навіть на зображеннях із 50% пошкодженості. Для тестування, було простежено зміни точності не на одному тестовому зображенні, а на всій вибірці із попереднього порівняння. Аналоги на залежність від рівня шуму не тестувалися через час обробки складних для розпізнавання зображень деякими з них.

В цілому, розроблена програма показала задовільні швидкість і точність роботи, низьку залежність результату від рівня шуму зображення, довела надійність роботи і конкурентоспроможність.

4.3 Висновки за розділом 4

Програма оснащена інтуїтивно зрозумілим і зручним графічним інтерфейсом, який забезпечує простоту та комфорт під час роботи користувача.

Розташування і кількість елементів інтерфейсу, із якими може взаємодіяти користувач, зведено до необхідного мінімуму, або потік роботи

програми максимально відповідав очікуванням користувача від того, що він натискає.

Після запуску застосунку користувач може завантажити необхідне зображення для подальшої обробки. У процесі виконання завдання на екрані відображається індикатор прогресу, що дозволяє відстежувати поточний стан операції. Після завершення обробки користувач має можливість видалити завантажений файл або скопіювати розпізнаний текст одним натисканням, а також бачить результат розпізнавання, який можна скопіювати при натисканні однієї кнопки.

Було проведено тестування застосунку і порівняння із аналогами, які виконують подібні завдання. Проєкт відзначається стабільною роботою, надійністю в експлуатації та повністю готовий до практичного використання. Крім того, його структура дозволяє легко адаптувати програму до нових вимог або інтегрувати з іншими програмними рішеннями. В порівнянні з аналогами, розроблений застосунок також показав конкурентоспроможний результат в усіх випробуваннях і довів свою актуальність.

ВИСНОВКИ

У результаті виконання магістерської роботи досягнуто всі поставлені цілі та виконано заплановані завдання. Основні етапи дослідження та їх результати викладено нижче.

Проведено аналіз методів розпізнавання рукописного тексту та шляху їх розвитку. Проведено детальне ознайомлення із предметною областю – розпізнаванням рукописного тексту, ознайомлення із найбільш важливими питаннями цієї проблемної області та проаналізовано існуючі рішення. Сформована постановка завдання до роботи і доведена можливість її здійснення. Також у першому розділі було доведено актуальність проєкту.

У другому розділі описано причину вибору технологій розробки, їх опис, переваги і недоліки.

У межах роботи мовою реалізації обрано Python, що зумовлено його широким застосуванням у галузі машинного навчання та наявністю потужного інструментарію для побудови і навчання нейронних мереж (TensorFlow, Keras, PyTorch). Для навчання нейронної мережі, було обрано платформу TensorFlow і, як середовище розробки, було використано Visual Studio Code.

У третьому розділі проведено аналіз теоретичних основ машинного навчання та глибинних нейронних мереж, які застосовуються для задач оптичного розпізнавання тексту. Розглянуто сучасні архітектури моделей, зокрема рекурентні (RNN, LSTM) та згорткові нейронні мережі (CNN), а також їх комбінації, які забезпечують високу точність під час розпізнавання послідовностей символів. Також було описано процес збору і обробки вхідних даних, а також процес тренування нейронної мережі.

Окрім нейронної мережі, у третьому розділі було спроектовано функціональну схему програми і Uml-діаграму композитної структури класів, які наглядно демонструють теоретичний та практичний потік роботи

при виконанні завдання. Описано кожен використаний метод, кожную модифікацію і причини їх застосування.

Четвертий розділ присвячено інтерфейсу і тестуванню програми.

Для підвищення зручності використання створено інтерактивний графічний інтерфейс, який дозволяє користувачу завантажувати зображення і переглядати результати розпізнавання з візуалізацією. Наведено потік функціонування програми і усі форми, які користувач може побачити.

Після проведення тестування, отримані результати підтверджують ефективність обраних методів і створюють основу для подальших досліджень у сфері інтелектуального аналізу зображень і документів. В порівнянні із аналогами, розроблена програма видає конкурентоспроможні результати. До того ж, розглянуто залежність результату роботи програми від пошкодженості вхідного зображення і втрати чіткості розпізнавання визначено як мінімальні і допустимі.

В подальшому можна модернізувати програмний продукт в напрямку розширення функціоналу, наприклад, шляхом додавання різних мов для розпізнавання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Maad M. M. Handwriting Recognition Methods / Maad M. M. // IEEE Transactions on Signal Processing. - 2005. - pp. 1-3.
2. Sardar Hasen Ali, Maiwan Bahjat Abdulrazzaq. A Comprehensive Overview of Handwritten Recognition Techniques: A Survey. Journal of Computer Science, Vol. 19, No. 5, 2023, pp. 569–587. DOI: 10.3844/jcssp.2023.569.587.
3. S. K. Singh, A. K. Singh, and M. M. Raghuwanshi, “Deep learning-based approaches for character and word segmentation in complex handwritten documents: A comprehensive survey,” Pattern Recognition Letters, vol. 172, pp. 33–47, 2023.
4. J. Zakraoui, M. Saleh, S. Al-Maadeed, and J. M. AlJa’am, “A study of children emotion and their performance while handwriting arabic characters using a haptic device,” Education and Information Technologies, vol. 28, no. 2, pp. 1783–1808, 2023.
5. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” Advances in Neural Information Processing Systems (NeurIPS), pp. 5998–6008, 2017.
6. Ingle, R. Reeve, Fujii, Yasuhisa, Deselaers, Thomas, Baccash, Jonathan, Popat, Ashok C. «A Scalable Handwritten Text Recognition System». arXiv preprint arXiv:1904.09150, 2019. URL: <https://arxiv.org/abs/1904.09150>
7. Amershi, Saleema et al. «Software Testing in the Age of AI: Challenges and Opportunities». IEEE Software, Vol. 35, No. 5, Sept/Oct 2018, pp. 40-49. DOI:10.1109/MS.2018.2901110.
8. Vinciarelli, A. A survey of off-line cursive script recognition // Pattern Recognition. – 2002. – Vol. 35, No. 7. – pp. 1433–1446.
9. Ruoxin Li. A review of neural networks in handwritten character recognition. Applied and Computational Engineering, Vol. 92, 2024, pp. 169–174.
10. Dhruv Shah, Sai Bhargav, Dhanush B. Bridging the Gap between Handwriting and Machine: AI-based Handwritten Text Recognition. International

Journal of Computer Applications, Vol. 185 No. 7, May 2023, pp. 39–45.
DOI: 10.5120/ijca2023922727.

11. Plamondon, R.; Srihari, S. N. On-line and off-line handwriting recognition: a comprehensive survey // IEEE Transactions on Pattern Analysis and Machine Intelligence. – 2000. – Vol. 22, No. 1. – pp. 63–84.

12. Булах, І. Є. Розпізнавання рукописного тексту на основі нейронних мереж // Вісник Київського національного університету імені Тараса Шевченка. Серія: Прикладна математика та інформатика. – 2017. – № 1. – С. 54–60.

13. Cho, K.; Van Merriënboer, B.; Bahdanau, D.; Bengio, Y. On the Properties of Neural Machine Translation: Encoder–Decoder Approaches // Proceedings of SSST-8: Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation. – 2014. – pp. 103–111.

14. Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” nature, vol. 521, no. 7553, p. 436, 2015.

15. P. Voigtlaender, M. Rottmann, and T. Dreier, “Comprehensive deep CNN architectures for document image understanding and text recognition,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 45, no. 3, pp. 1238–1252, 2023.

16. N. Boufenar and M. Batouche, “Deep hybrid HMM-CRNN architecture for unconstrained handwritten text recognition,” Neural Computing and Applications, vol. 35, pp. 12401–12415, 2023.

17. Terras, M. “Digital Curations: Notes on Handwriting Recognition and the Transkribus Platform” // Journal of Documentation. – 2017. – Vol. 73, No. 5. – pp. 855–871.

18. Qi, G.; Lin, F. “Evaluating Google Cloud Vision for Document Image Understanding” // International Journal of Advanced Computer Science and Applications. – 2017. – Vol. 8, No. 12. – pp. 90–96.

19. Smith, R. “An Overview of the Tesseract OCR Engine” // Proceedings of the Ninth International Conference on Document Analysis and Recognition (ICDAR). – 2007. – Vol. 2. – pp. 629–633.
20. Graves, A.; Liwicki, M.; Fernández, S.; Bertolami, R.; Bunke, H.; Schmidhuber, J. A Novel Connectionist System for Unconstrained Handwriting Recognition // IEEE Transactions on Pattern Analysis and Machine Intelligence. – 2009. – Vol. 31, No. 5. – pp. 855–868.
21. Patil, A.; Vora, M. “A Comparative Study of Cloud-Based Machine Learning Services: Amazon Machine Learning and Others” // International Journal of Computer Applications. – 2016. – Vol. 144, No. 4. – pp. 1–7.
22. Barga, R.; Goldstein, J.; Larsen, M. “Introducing Microsoft Azure Machine Learning” // Microsoft Research Technical Report. – 2014. – pp. 1–12.
23. Gautam, M. T. K.; Goel, O.; Jindal, R.; Gupta, P. «Handwritten Character Recognition Using TensorFlow and CNN». Journal of Quantum Science and Technology, Vol. 1, No. 4, Nov 2024, pp. 572-577. DOI: 10.63345/jqst.v1i4.137.
24. Josephine Prem Kumar, Dharshan H. D. Handwritten Text Recognition Using Deep Learning: A CNN-LSTM Approach. International Journal of Intelligent Systems and Applications in Engineering, Vol. 13 No. 1, 2025, pp. 14–22.
25. Hochreiter, S.; Schmidhuber, J. “Long Short-Term Memory” // Neural Computation. – 1997. – Vol. 9, No. 8. – pp. 1735–1780.

ДОДАТОК А
Фрагмент тексту програми

A.1 Реалізація BOX алгоритму

```

import cv2
import numpy as np
from matplotlib import pyplot as plt
import sys
import os

def main(image_path, ksize=(5, 5)):
    if not os.path.exists(image_path):
        raise FileNotFoundError(f"Файл '{image_path}' не знайдено.")
    if not isinstance(ksize, tuple) or len(ksize) != 2 or not all(isinstance
(x, int) and x > 0 for x in ksize):
        raise ValueError("ksize має бути кортежем з двох додатних цілих
чисел.")
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if image is None:
        raise ValueError("Не вдалося завантажити зображення або файл не
є зображенням.")
    try:
        blurred = cv2.blur(image, ksize=ksize)
    except Exception as e:
        raise RuntimeError(f"Помилка під час застосування
BOX-фільтра: {e}")
    fig, axes = plt.subplots(1, 2, figsize=(10, 5))
    for ax, img, title in zip(axes, [image, blurred], ['Оригінал',
'BOX-фільтр']):
        ax.imshow(img, cmap='gray')
        ax.set_title(title)
        ax.axis('off')
    plt.tight_layout()
    plt.show()

if __name__ == "__main__":
    try:
        path = sys.argv[1] if len(sys.argv) > 1 else 'image.jpg'
        main(path, ksize=(5, 5))
    except Exception as err:
        print(f"Помилка: {err}")
        sys.exit(1)

```

A.2 Реалізація алгоритму Canny Edge Detection

```

import cv2
import numpy as np
from matplotlib import pyplot as plt
import os
import sys

def non_max_suppression(M, theta):
    if M is None or theta is None:
        raise ValueError("Аргументи M і theta не можуть бути None.")
    if M.shape != theta.shape:
        raise ValueError("Розміри M і theta повинні збігатися.")
    Z = np.zeros_like(M)
    angle = theta * 180. / np.pi; angle[angle < 0] += 180
    rows, cols = M.shape
    for i in range(1, rows - 1):
        for j in range(1, cols - 1):
            try:
                q, r = 255, 255
                if (0 <= angle[i, j] < 22.5) or (157.5 <= angle[i, j] <= 180):
                    q, r = M[i, j + 1], M[i, j - 1]
                elif (22.5 <= angle[i, j] < 67.5):
                    q, r = M[i + 1, j - 1], M[i - 1, j + 1]
                elif (67.5 <= angle[i, j] < 112.5):
                    q, r = M[i + 1, j], M[i - 1, j]
                elif (112.5 <= angle[i, j] < 157.5):
                    q, r = M[i - 1, j - 1], M[i + 1, j + 1]
                Z[i, j] = M[i, j] if (M[i, j] >= q and M[i, j] >= r) else 0
            except IndexError:
                continue
    return Z

def hysteresis_thresholding(img, lowThreshold, highThreshold):
    if not isinstance(lowThreshold, (int, float)) or not
isinstance(highThreshold, (int, float)):
        raise ValueError("Пороги мають бути числовими значеннями.")
    if lowThreshold >= highThreshold:
        raise ValueError("lowThreshold повинен бути меншим за
highThreshold."); strong, weak = 255, 75
    result = np.zeros_like(img)
    strong_i, strong_j = np.where(img >= highThreshold)
    weak_i, weak_j = np.where((img <= highThreshold) & (img >=
lowThreshold))

```

```

result[strong_i, strong_j] = strong
result[weak_i, weak_j] = weak
rows, cols = img.shape
for i in range(1, rows - 1):
    for j in range(1, cols - 1):
        if result[i, j] == weak:
            if np.any(result[i - 1:i + 2, j - 1:j + 2] == strong):
                result[i, j] = strong
return result

def main(image_path):
    if not os.path.exists(image_path):
        raise FileNotFoundError(f'Файл '{image_path}' не знайдено.')
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if image is None:
        raise ValueError("Не вдалося завантажити зображення або файл не є зображенням.")
    try:
        blurred = cv2.GaussianBlur(image, (5, 5), sigmaX=1.4)
    except Exception as e:
        raise RuntimeError(f"Помилка при згладжуванні: {e}")
    I_x = cv2.Sobel(blurred, cv2.CV_64F, 1, 0, ksize=3)
    I_y = cv2.Sobel(blurred, cv2.CV_64F, 0, 1, ksize=3)
    M = np.sqrt(I_x ** 2 + I_y ** 2)
    theta = np.arctan2(I_y, I_x)
    M_suppressed = non_max_suppression(M, theta)
    edges = hysteresis_thresholding(M_suppressed, lowThreshold=50,
highThreshold=100)
    fig, axes = plt.subplots(1, 2, figsize=(10, 5))
    for ax, img, title in zip(axes, [image, edges], ['Оригінал', 'Контури (Canny)']):
        ax.imshow(img, cmap='gray')
        ax.set_title(title); ax.axis('off')
    plt.tight_layout()
    plt.show()

if __name__ == "__main__":
    try:
        path = sys.argv[1] if len(sys.argv) > 1 else 'image.jpg'
        main(path)
    except Exception as err:
        print(f"Помилка: {err}")
        sys.exit(1)

```

ДОДАТОК Б
Слайди презентації

РОЗРОБКА ТА ДОСЛІДЖЕННЯ АЛГОРИТМІВ МАШИННОГО НАВЧАННЯ ДЛЯ РОЗПІЗНАВАННЯ РУКОПИСНОГО ТЕКСТУ

Виконав:
ст. групи КНТ 124-м

Геннадій НЕСТЕРОВ

Керівник:
к.т.н., доцент

Євгеній ГОФМАН

Рисунок Б.1 – Слайд 1

ОБ'ЄКТ, ПРЕДМЕТ ТА МЕТА РОБОТИ

- Об'єкт дослідження – процес розробки програмного забезпечення для розпізнавання рукописного тексту.
- Предмет дослідження – програмні засоби для реалізації алгоритмів машинного навчання для розпізнавання рукописного тексту.
- Мета роботи – створення програмного забезпечення для підвищення точності розпізнавання рукописного тексту.

Рисунок Б.2 – Слайд 2

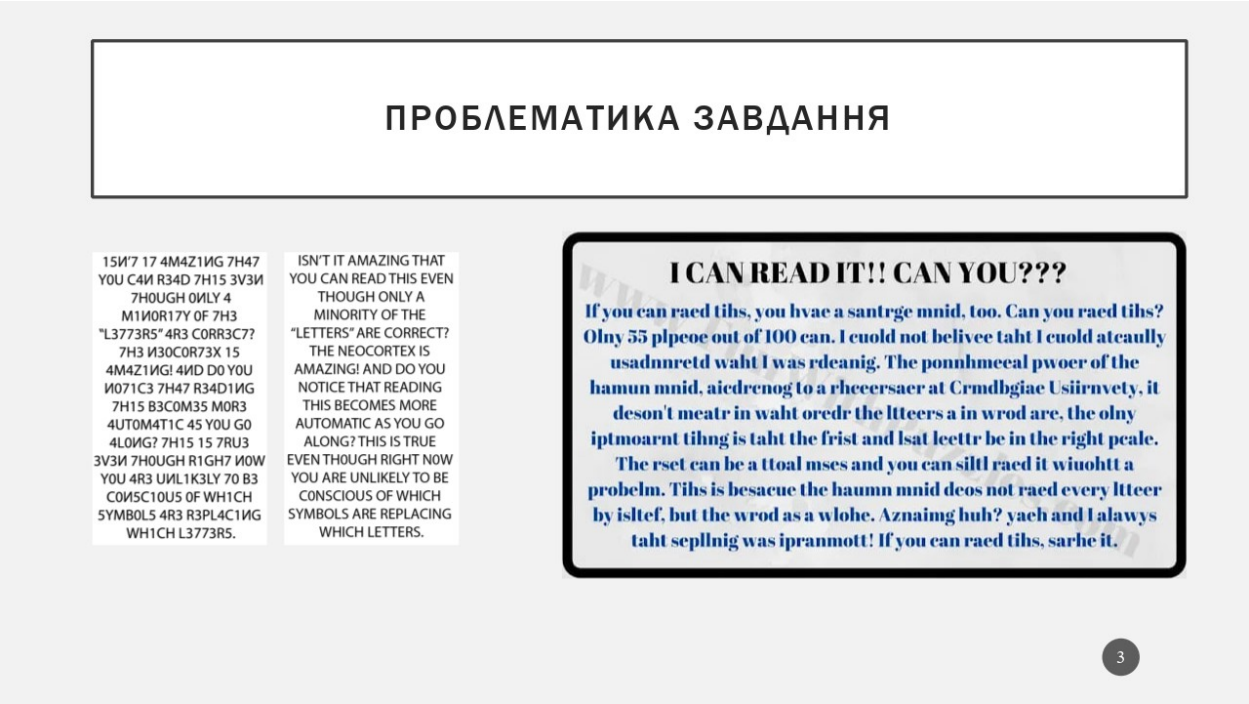


Рисунок Б.3 – Слайд 3

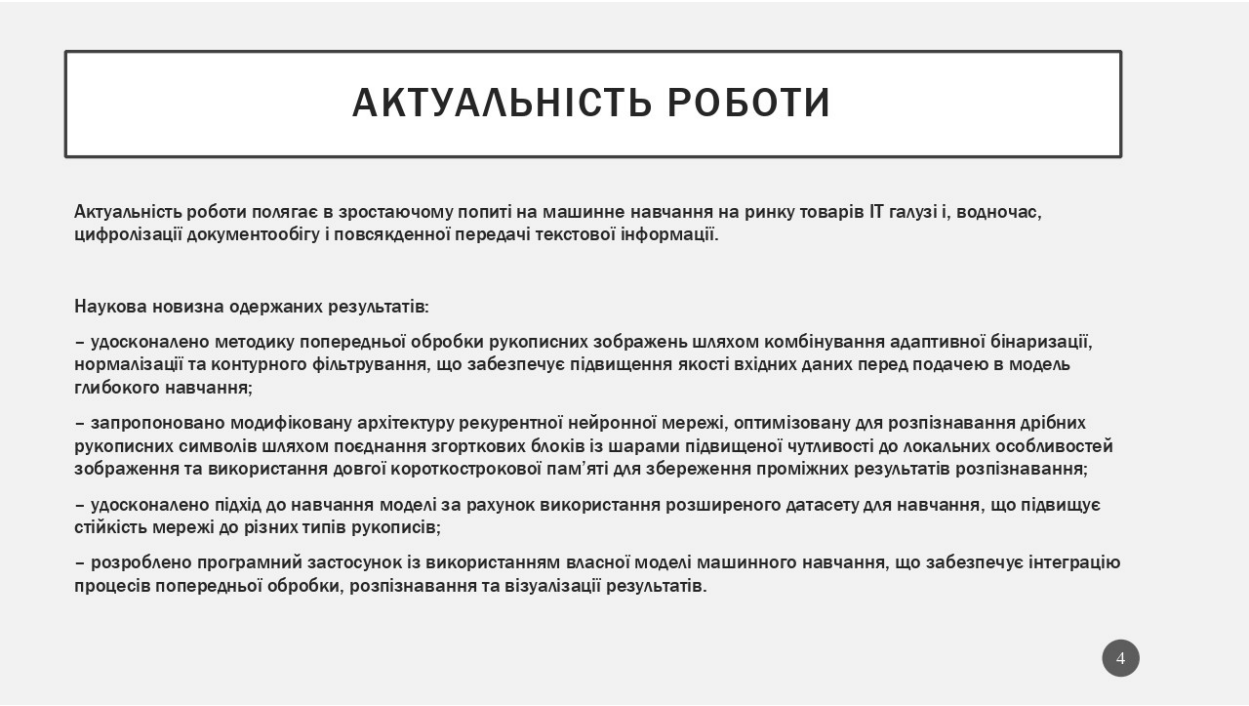


Рисунок Б.4 – Слайд 4

ПОСТАНОВКА ЗАВДАННЯ

$$\text{Текст} = \arg \max \text{ CTC } (\text{LSTM } (\text{CNN } (\text{Зображення})))$$

- 1.Зображення: На вхід подається фотографія або скан рукописного тексту.
- 2.CNN (згортова нейронна мережа): Виділяє візуальні ознаки: лінії, криві, частини літер.
- 3.LSTM (рекурентна нейронна мережа): Аналізує ці ознаки зліва направо та враховує порядок символів у слові.
- 4.CTC (алгоритм зчитування): Перетворює послідовність імовірних символів у зрозумілий текст, навіть якщо літери не відокремлені чітко.
- 5.arg max: Обирає текст, який мережа вважає найбільш імовірним.

5

Рисунок Б.5 – Слайд 5

ВИБІР МОВИ ПРОГРАМУВАННЯ

Критерій	Python	Java	Scala	R
Швидкість роботи	±	+	±	±
Бібліотеки	++	±	±	+
Зручність роботи з нейронними мережами	++	+	+	+
Спільнота	++	+	±	+

6

Рисунок Б.6 – Слайд 6

ПОРІВНЯННЯ ФРЕЙМВОРКІВ МАШИННОГО НАВЧАННЯ

Критерій	Amazon Machine Learning	Azure Machine Learning Studio	TensorFlow
Підтримка нейронних мереж	+	+	++
Гнучкість архітектур	-	+	++
Підтримка CNN + LSTM	+	-	+
Продуктивність навчання	-	+	++
Зручність використання	+	++	+
Придатність для OCR	-	+	++

7

Рисунок Б.7 – Слайд 7

ПОРІВНЯННЯ СЕРЕДОВИЩ РОЗРОБКИ

Критерій	PyCharm	Jupyter Notebook	Spyder	Visual Studio Code
Наявність безкоштовної версії	++	++	++	++
Підтримка TensorFlow	++	+	+	++
Кросплатформність	++	++	++	++
Швидкість роботи	±	++	+	++
Підтримка та оновлення	++	+	+	++

8

Рисунок Б.8 – Слайд 8



Рисунок Б.9 – Слайд 9

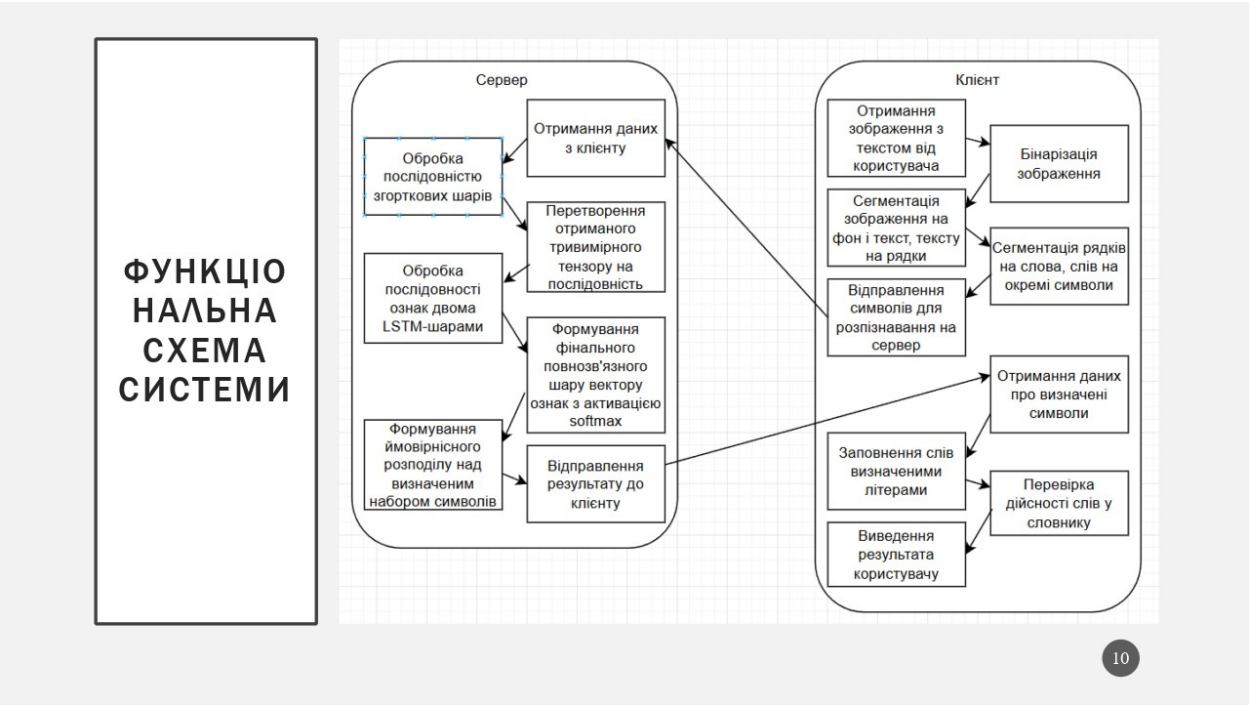


Рисунок Б.10 – Слайд 10



Рисунок Б.11 – Слайд 11

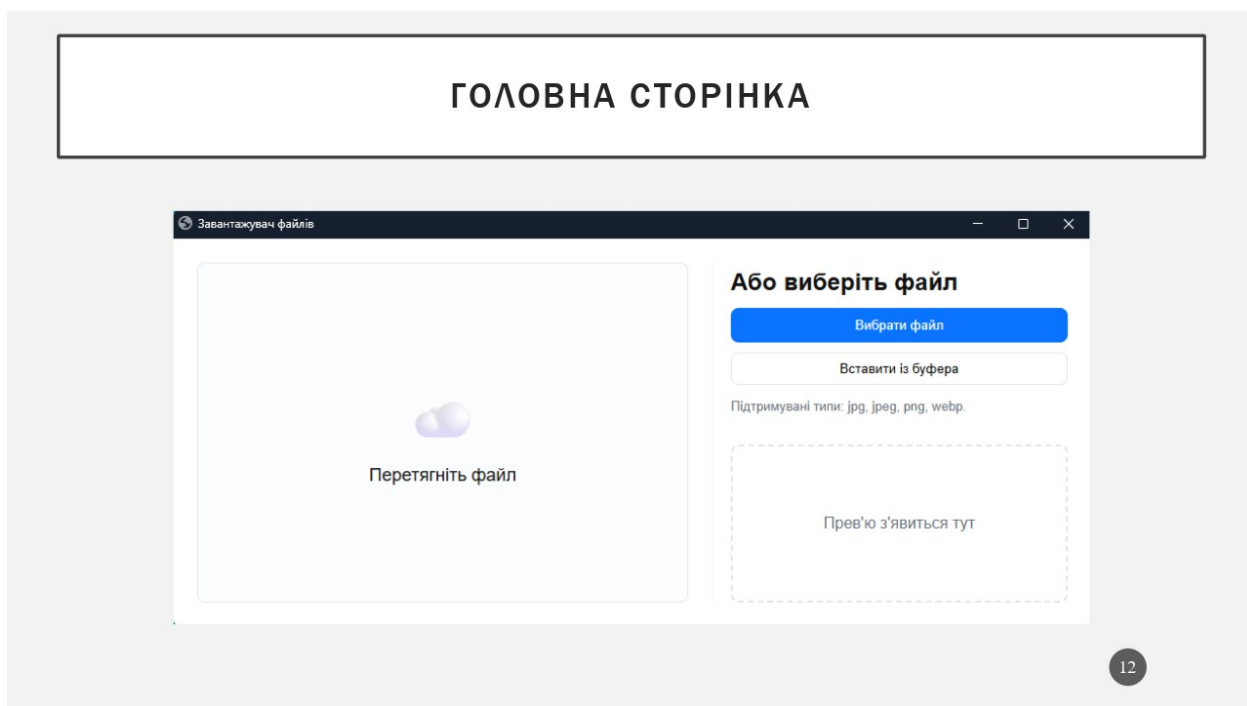
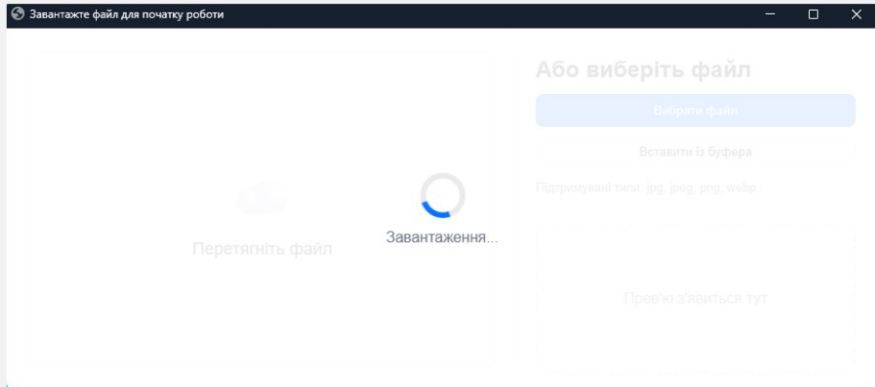


Рисунок Б.12 – Слайд 12

ФРЕЙМ ЗАВАНТАЖЕННЯ РЕЗУЛЬТАТУ



13

Рисунок Б.13 – Слайд 13

ВИВЕДЕННЯ РЕЗУЛЬТАТУ

Привіт!

Мені дуже подобається, що ти зараз читаєш мого лист. Хочеться, щоб цей звичайний набір слів добре, був дійсно щасливим!

Життєво-чудовий і надзвичайний, неповторний і особливий з якими зустрівся, ще раз доводити!

Навіть погляди, жести після того як не потворні, особливі.

Дорогий Друг, пам'яті що найкраще в цьому житті для тебе!

Люби, радій.

З повагою
Л.М.



14

Рисунок Б.14 – Слайд 14

ВИСНОВКИ

В результаті виконаної роботи було проаналізовано та досліджено засоби розпізнавання рукописного тексту і саму предметну область. Сформульовано функціональні вимоги до програмного забезпечення, обраного середовища розробки і функціонування додатку, а також розглянуто інструментарій розробки.

Розроблено програмне забезпечення для вирішення розглянутої задачі. Запропоновано структуру, підхід та описано основні модулі програмного забезпечення, а потім реалізовано новий метод вирішення задачі. Написані алгоритми передобробки зображення, натреновано нейронну мережу для його розпізнавання.

Виконано тестування розробленого програмного забезпечення та доведено, що розроблене програмне забезпечення є працездатним, достатньо точним у розпізнаванні і виконує усі заявлені функції. За результатами тестування програмного забезпечення не виявлено помилок, збоїв або інших серйозних проблем, тому можна зробити висновок, що тестування було успішним, а програмне забезпечення готове до використання.

15

Рисунок Б.15 – Слайд 15

ДЯКУЮ ЗА УВАГУ

Рисунок Б.16 – Слайд 16