

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 проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ
ЗАСОБІВ РОЗПІЗНАВАННЯ РУКОПИСНОГО ТЕКСТУ
RESEARCH AND SOFTWARE IMPLEMENTATION OF
HANDWRITING RECOGNITION TOOLS

Виконав(ла): студент(ка) 2 курсу, групи КНТ-114м
Спеціальності 121 Інженерія програмного
(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ПАВЛЕЦЬ О.Д.

(ПРІЗВИЩЕ та ініціали)

Керівник СКРУПСЬКИЙ С.Ю.

(ПРІЗВИЩЕ та ініціали)

Рецензент КОЦУР М.І.

(ПРІЗВИЩЕ 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)

ПАВЛЕЦЯ Олексія Дмитровича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація засобів розпізнавання рукописного тексту. Research and Software Implementation of Handwriting Recognition Tools

керівник проєкту (роботи) к.т.н., доцент, СКРУПСЬКИЙ Степан Юрійович,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

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

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

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Матеріали і методи. 3. Опис програми. 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	СКРУПСЬКИЙ С.Ю., 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.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка структури програми.	3 тиждень	Розділ 3
5	Розробка програми.	4-7 тижні	Розділи 3,4
6	Тестування та експериментальне дослідження програмного забезпечення.	8 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	9-10 тижні	Додатки
8	Нормоконтроль та рецензування.	11 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка)

(підпис) Олексій ПАВЛЕЦЬ
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Степан СКРУПСЬКИЙ
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
85 с., 3 табл., 28 рис., 3 дод., 18 джерел.

PYTHON, VISUAL STUDIO CODE, ЗАСТОСУНОК, МОВА ПРОГРАМУВАННЯ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, РОЗПІЗНАВАННЯ, РУКОПИСНИЙ ТЕКСТ.

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

Предмет дослідження – програмні засоби для розпізнавання рукописного тексту.

Мета роботи – дослідження та розробка програмного забезпечення для розпізнавання рукописного тексту.

Матеріали, методи та технічні засоби: мова програмування Python, середовище розробки Visual Studio Code.

Результати. Розроблено програмне забезпечення для розпізнавання рукописного тексту за допомогою мови програмування Python та середовища розробки Visual Studio Code.

Практична цінність роботи полягає у розробці програмного забезпечення для розпізнавання рукописного тексту.

Висновки. Виконано проєктування програмного забезпечення для розпізнавання рукописного тексту. Розроблено програмне забезпечення для розпізнавання рукописного тексту. Здійснено тестування розробленого програмного забезпечення для розпізнавання рукописного тексту.

Галузь використання – розпізнавання рукописного тексту.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 85 pages, 3 tables, 28 figures, 3 appendixes, 18 sources.

PYTHON, VISUAL STUDIO CODE, APPLICATION, PROGRAMMING LANGUAGE, SOFTWARE, RECOGNITION, HANDWRITTEN TEXT.

The object of research is the process of developing software for handwriting recognition.

The subject of the research is software for handwriting recognition.

The purpose of this work is to develop the software for handwriting recognition.

Materials, methods and technical tools: Python programming language, Visual Studio Code development environment.

Results. The software tools for handwriting recognition using the Python programming language and the Visual Studio Code development environment have been developed.

The practical value of the work lies in the development of software for handwriting recognition.

Conclusions. The software for handwriting recognition has been designed. The software for handwriting recognition has been developed. The developed software for handwriting recognition has been tested.

Scope of use – handwriting recognition.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок.....	8
Вступ	9
1 Аналіз предметної області	11
1.1 Програмне забезпечення для розпізнавання рукописного тексту	11
1.2 Аналіз програмного забезпечення для розпізнавання рукописного тексту	20
1.3 Висновки за розділом 1	29
2 Матеріали і методи.....	30
2.1 Вибір мови програмування	30
2.2 Вибір середовища розробки для створення програмного забезпечення для розпізнавання рукописного тексту	33
2.3 Розробка моделі взаємодії користувача з програмним забезпеченням для розпізнавання рукописного тексту	36
2.4 Висновки за розділом 2	38
3 Опис програми.....	39
3.1 Структура програмного забезпечення для розпізнавання рукописного тексту	39
3.2 Функціонування програмного забезпечення для розпізнавання рукописного тексту.....	41
3.3 Синтез нейромережевої моделі для розпізнавання рукописного тексту	44
3.4 Проєктування інтерфейсу програмного забезпечення для розпізнавання рукописного тексту	47
3.5 Висновки за розділом 3	49
4 Експлуатація, тестування та експериментальне дослідження програми	51
4.1 Призначення й умови застосування програми	51
4.2 Характеристики програми для розпізнавання рукописного тексту	52
4.3 Інструкція по експлуатації програми.....	53
4.3.1 Звернення до програми.....	53

4.3.2 Вхідні й вихідні дані	53
4.3.3 Повідомлення	53
4.4 Виконання програмного забезпечення для розпізнавання рукописного тексту	54
4.5 Тестування програмного забезпечення для розпізнавання рукописного тексту	56
4.6 Висновки за розділом 4	57
Висновки	58
Перелік джерел посилань	61
Додаток А Технічне завдання	63
Додаток Б Фрагмент тексту програми	67
Додаток В Слайди презентації	79

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

CNN – Convolutional Neural Network;

CTC – Connectionist Temporal Classification;

HR – Handwriting Recognition;

LSTM – Long Short Term Memory;

ПЗ – програмне забезпечення;

ШІ – штучний інтелект.

ВСТУП

Актуальність програмного забезпечення для розпізнавання рукописного тексту зумовлюється інтенсивними процесами цифрової трансформації суспільства та зростанням потреби у швидкому перетворенні аналогової інформації на електронний формат. Незважаючи на широке впровадження електронних форм введення даних, значна частина інформації продовжує фіксуватися у рукописному вигляді, що характерно для освітньої, адміністративної, юридичної, медичної та архівної сфер. Оцифрування таких даних дає змогу суттєво скоротити витрати часу на їх опрацювання, забезпечити зручний пошук, зберігання та інтеграцію у сучасні інформаційні системи [1], [2].

Розвиток технологій штучного інтелекту (ШІ) та методів глибинного навчання сприяв значному підвищенню точності алгоритмів оптичного та рукописного розпізнавання, що уможливило їх ефективне використання навіть за наявності нерозбірливого, стилістично змішаного чи індивідуалізованого письма. Така здатність відкрила перспективи автоматизації процесів, які раніше вимагали виключно ручної праці [3], [4].

Додатковим чинником, що підсилює значення даного напрямку, є активне впровадження цифрових рішень у сферу збереження культурної спадщини, де автоматизоване розпізнавання текстів сприяє відновленню та упорядкуванню архівних документів, рукописних пам'яток та історичних матеріалів. Паралельно розвиток мобільних додатків та хмарних сервісів зробив цю технологію доступною не лише для спеціалізованих установ, а й для широкого кола користувачів, що зумовлює її інтеграцію у повсякденну практику [3]-[6].

Проте деякі програми для розпізнавання рукописного тексту можуть використовувати технології штучного інтелекту та глибинного навчання, що є чутливими до нерозбірливого почерку, стилістичних відмінностей у написанні символів та індивідуальних особливостей письма. Значний вплив на точність

результатів справляють фактори, пов'язані з якістю вхідних даних, зокрема наявність шумів, викривлень або низька роздільна здатність зображення. Певну проблему становить і коректне розпізнавання текстів, написаних у змішаних форматах, де використовуються різні мови, алфавіти або шрифти, що вимагає додаткових обчислювальних ресурсів та налаштувань моделі. Окремою проблемою є розпізнавання спеціалізованих позначень, формул чи скорочень, що часто зустрічаються у технічних, наукових або медичних записах. Застосування таких програмних рішень у реальному середовищі може бути обмежене високими вимогами до обчислювальної потужності, потребою у великих обсягах навчальних даних та необхідністю періодичного оновлення моделей [1]-[6].

Тому актуальною є розробка програмного забезпечення для розпізнавання рукописного тексту. У дипломній кваліфікаційній роботі магістра розв'язується актуальне завдання розробки та дослідження програмного забезпечення для розпізнавання рукописного тексту.

Для досягнення поставленої мети у кваліфікаційній роботі магістра необхідно розв'язати такі задачі:

- виконати аналіз предметної області та програмних засобів для розпізнавання рукописного тексту;
- розробити модель взаємодії користувача з програмним забезпеченням для розпізнавання рукописного тексту;
- здійснити проектування програмного забезпечення для розпізнавання рукописного тексту;
- створити програмне забезпечення для розпізнавання рукописного тексту;
- виконати тестування розробленого програмного забезпечення для розпізнавання рукописного тексту.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Програмне забезпечення для розпізнавання рукописного тексту

Розпізнавання рукописного тексту являє собою складну задачу, оскільки почерк людини відзначається великою варіативністю, нерівномірністю символів та їх накладанням. Для забезпечення точності програмні рішення в цій сфері використовують методи штучного інтелекту, зокрема нейронні мережі, здатні навчатися на великих масивах даних і виявляти закономірності навіть у складних почерках. Основна перевага подібних програмних систем полягає у можливості швидкого перетворення рукописних записів у цифрову форму, що значно полегшує обробку та зберігання інформації. Додатково до цього створюються умови для автоматизації процесів у сферах освіти, медицини чи діловодства, де часто доводиться працювати з великою кількістю рукописних документів [1].

Разом з тим, складність завдання проявляється у високій залежності точності від якості даних, якими навчається модель. Нерозбірливий почерк, помилки у написанні чи поєднання різних мовних символів можуть призводити до помилок у результатах. До недоліків також належить необхідність значних обчислювальних ресурсів під час тренування моделей та висока вартість розробки таких рішень [1].

Створення програмного забезпечення для розпізнавання рукописного тексту зазвичай починається з етапу збору великих масивів зразків написаного тексту, які використовуються для навчання алгоритмів. Далі відбувається попередня обробка даних, що включає нормалізацію зображень, фільтрацію шумів та сегментацію символів. Після цього моделі навчаються виявляти характерні ознаки букв і слів. На завершальному етапі результати проходять перевірку та оптимізацію, аби забезпечити стабільну роботу системи в реальних умовах [1].

Сучасні системи оптичного розпізнавання друкованого тексту досягають дуже високої точності, проте при роботі з рукописними записами ситуація значно складніша. Різні стилі письма, індивідуальні особливості почерку, з'єднання символів та їх нерівномірність ускладнюють процес автоматичного відтворення тексту. Тому результативність розпізнавання рукописних документів завжди нижча, ніж у випадку зі стандартним друкованим шрифтом [1].

Під рукописним введенням розуміють як друковані рукописні символи, так і скоропис, де букви зливаються між собою. Перший варіант інструменти здатні ідентифікувати більш надійно, тоді як другий потребує складніших алгоритмів аналізу. Окрему категорію становить цифровий рукописний текст, який фіксується на екранах планшетів чи смартфонів, що дозволяє системам розпізнавання враховувати динаміку рухів користувача й досягати точніших результатів [1].

Основними перешкодами в перетворенні рукописного вводу на структурований текст залишаються низька якість зображень, фонові шуми, різноманіття алфавітів та стилів письма, а також випадки злиття чи викривлення символів. Подолати ці труднощі можливо завдяки використанню глибоких нейронних мереж, методів сегментації та алгоритмів попередньої обробки, які прибирають фон і покращують контрастність [1].

Щоб підвищити точність перетворення, рекомендується готувати документи з урахуванням технічних вимог: забезпечувати чіткість написання, уникати кольорового фону, дотримуватись структури з чіткими полями та елементами вирівнювання. Додатково використовуються допоміжні елементи, наприклад штрих-коди чи прапорці, які спрощують інтерпретацію даних і зменшують ризик помилок [1].

Активні дослідження у сфері розпізнавання рукописного тексту пов'язані з розвитком глибинного навчання. Алгоритми навчаються на величезних корпусах рукописних символів, а також використовують мовні моделі для корекції результатів. Це дозволяє враховувати контекст і підбирати

найбільш ймовірні варіанти, якщо окремий символ розпізнано невпевнено. При виборі інструментів для розпізнавання рукописного тексту важливими залишаються такі чинники, як точність, швидкість роботи, здатність алгоритмів постійно вдосконалюватися, а також зручність взаємодії з користувачем. Значну роль відіграють вимоги безпеки, масштабованість рішень та гнучкість їхнього застосування в різних сферах [1]-[3].

Розпізнавання рукописного тексту набуває практичного значення в різних сферах, де обіг інформації все ще значною мірою пов'язаний із письмовими документами. У фармацевтичній практиці рецептурні бланки залишаються джерелом труднощів: кожен почерк має власні особливості, і помилка при ідентифікації символів може вплинути на якість медичної допомоги. У страховій сфері тисячі рукописних форм і заявок щодня проходять обробку, де неточне зчитування даних призводить до затримок або невірних рішень щодо компенсацій. Аналогічні проблеми існують у банківській галузі, де обробка чеків усе ще зберігає свою актуальність: значні обсяги документів потребують автоматизації, оскільки ручне введення уповільнює операційні процеси та створює додаткові витрати. У бібліотечних і архівних проєктах цифровізація рукописних матеріалів є необхідною умовою для їхнього повноцінного використання: без перетворення зображень у структурований текст пошук, індексація та аналіз стають малоефективними [1]-[3].

Методи, що застосовуються для цієї задачі, поділяються на онлайн і офлайн підходи. У першому випадку використовується динаміка письма – траєкторії, напрямки та натиск пера чи стилуса, що дає змогу з високою точністю відокремлювати символи й реконструювати текст. У другому варіанті аналіз здійснюється вже після створення документа, коли доступним є лише його зображення. Такий підхід є більш поширеним, оскільки використання спеціальних цифрових інструментів не завжди можливе в масштабах реального життя [2], [3].

Початкові спроби автоматизації процесу базувалися на класичних методах машинного навчання, які потребували ручного виділення ознак символів. Це обмежувало гнучкість систем і знижувало їхню здатність адаптуватися до різних почерків. Значний прогрес став можливим завдяки застосуванню глибоких нейронних мереж. Багатовимірні рекурентні архітектури дозволяють враховувати контекст одночасно по горизонталі та вертикалі зображення, завдяки чому досягається більш повне розуміння структури рукопису. Для вирівнювання між зображенням і текстовим результатом застосовується спеціальний алгоритм, який усуває дублювання символів і забезпечує адекватне формування слів [2], [3].

Точність таких систем додатково зростає завдяки використанню мовних моделей, що допомагають коригувати результат, підбираючи найбільш імовірні варіанти на основі контексту. Це особливо важливо у випадках, коли окремі символи залишаються нечіткими чи неоднозначними. Сучасні методи розпізнавання поступово долають межі, які ще донедавна здавалося неможливо подолати, і відкривають нові можливості для автоматизації у сфері медицини, фінансів, страхування та збереження культурної спадщини [2], [3].

У сфері розпізнавання рукописного тексту значне поширення отримали архітектури, що поєднують кодувальники та декодувальники. Вони добре зарекомендували себе у суміжних завданнях, а їх розширення завдяки механізмам уваги дозволило покращити точність трансформації візуальної інформації у текст. Дослідження у цьому напрямі показали, що увага здатна компенсувати недоліки традиційної сегментації, оскільки забезпечує роботу з повною сторінкою без попереднього поділу на окремі рядки [2], [3].

Інший підхід, відомий як «згортка, увага та правопис», пропонує комбінацію згорткових нейронних мереж і двонаправлених для вилучення ознак, на які накладається механізм уваги. Декодер у цьому випадку формує текстову послідовність по символах, спираючись на контекст і релевантні вектори ознак. У моделі випробувано два варіанти уваги, контентний та

позиційний, що дає змогу точніше визначати повторювані елементи й уникати плутанини у прогнозах [2], [3].

Попри успішні результати, деякі нейромережеві архітектури мають певні обмеження, зокрема труднощі з паралельним навчанням. Це стало поштовхом до використання трансформаторів, які завдяки механізму самоуваги демонструють кращу масштабованість і забезпечують одночасну обробку різних частин вхідних даних. Такий підхід дозволяє кодувати інформацію на рівні символів, інтегрувати мовні закономірності без окремих мовних моделей і робити точніші передбачення навіть у випадку символів, відсутніх у словнику. Архітектура поєднує візуальний кодер, що витягує ознаки зображення, та текстовий транскриптор, який накладає мовну самоувагу і синхронізує результати з візуальними характеристиками через модуль взаємної уваги [2], [3].

Окремим напрямом розвитку стала генерація синтетичних зразків рукописів, що покликана компенсувати нестачу позначених даних. Відповідні нейромережеві архітектури здатні створювати різноманітні варіації текстових зображень, зберігаючи як зміст, так і стилістичні особливості. Такі моделі не потребують ретельної розмітки на рівні символів і дозволяють гнучко формувати корпуси для навчання, що є особливо важливим для багатомовних систем [2], [3].

Технології розпізнавання рукописного тексту сьогодні розглядаються як одна з найдинамічніших сфер розвитку штучного інтелекту. Якщо перетворення друкованих документів у цифрову форму за допомогою відповідних програмних систем уже стало повсякденною практикою, то робота з рукописом залишається значно складнішим завданням. Нерівномірність почерку, індивідуальні відмінності у написанні символів та комбінація різних мовних систем роблять цю проблему особливо актуальною [3], [4].

Завдяки поширенню мобільних пристроїв і зростанню обсягів інформації виникає потреба у швидкому й точному розпізнаванні рукописних

даних. Очікується, що впровадження таких технологій у сфері бізнесу, медицини, освіти та логістики стане ще масштабнішим. Уже зараз відчутно, як автоматизація процесів обробки записів дозволяє зменшувати витрати часу й ресурсів [3], [4].

Сутність розпізнавання рукописного тексту полягає у здатності програмних систем аналізувати як відскановані документи, так і цифрові записи, зроблені за допомогою спеціальних пристроїв. Для досягнення високої точності застосовуються алгоритми машинного навчання, здатні не лише виділяти окремі символи, але й враховувати контекст, перетворюючи їх у цілісні слова й речення [3], [4].

Очевидними перевагами цієї технології є оптимізація зберігання інформації, полегшений пошук даних, а також створення умов для більшої доступності контенту. Оцифровані документи займають менше місця, швидше обробляються й забезпечують безпечніший доступ, ніж традиційні паперові архіви. Саме тому банківські установи, лікарні та страхові компанії поступово переходять до електронних форматів, мінімізуючи ризики, пов'язані з використанням фізичних носіїв [3], [4].

Разом із тим існують проблеми, які сповільнюють розвиток цього напрямку. Висока мінливість почерку, низька якість зображень і складність уніфікації різних мовних моделей значно знижують точність систем. Навіть сучасні алгоритми часто потребують донавчання й постійної адаптації [3], [4].

Для вирішення цих завдань використовуються різні архітектури нейронних мереж. Серед них варто відзначити капсульні мережі, багатовимірні рекурентні структури та вдосконалені згорткові нейронні мережі (Convolutional Neural Network, CNN), які дозволяють не лише розпізнавати символи, а й моделювати складні просторові та контекстуальні зв'язки. Подібні рішення наближають створення систем, здатних працювати з будь-якими варіантами почерку, зберігаючи високу точність і швидкість обробки [3], [4].

При роботі з рукописним текстом головна проблема полягає у відсутності інформації про те, який саме сегмент зображення відповідає певному символу. Просте припущення про рівномірний розподіл простору не дає задовільного результату, адже почерк відзначається високою варіативністю. Часова класифікація долає цю складність шляхом підрахунку ймовірностей усіх можливих відповідностей між послідовністю вхідних ознак і символами, що утворюють текст [3], [4].

Рекурентні нейронні мережі найчастіше використовуються як базова архітектура для цього підходу. Вони дозволяють моделі враховувати контекст у часовому вимірі, формуючи матрицю ймовірностей для кожної часової позначки. Під час декодування може застосовуватися стратегія вибору найімовірнішого шляху або ж променевий пошук, де розглядається кілька кандидатів з найбільшою вірогідністю. У другому випадку якість результатів зростає, особливо у поєднанні з мовними моделями [3]-[5].

Попри зручність рекурентних нейронних мереж, їхня архітектура ускладнює паралельну обробку даних, що стає проблемою для великих обсягів інформації. Для подолання цього обмеження пропонуються трансформерні моделі. Завдяки механізму самоуваги вони здатні аналізувати всю послідовність одночасно, зберігаючи залежності між символами без послідовної обробки. Цей підхід дозволяє не лише підвищити швидкість, а й усунути необхідність додаткових мовних моделей, адже знання контексту інтегруються безпосередньо у саму архітектуру [3]-[5].

Складність створення систем розпізнавання рукопису пов'язана не лише з архітектурою, а й із браком різноманітних навчальних даних. Кожна мова і навіть кожен індивідуальний почерк мають свої особливості, тому побудова універсальних наборів даних залишається проблемною. Для її часткового вирішення використовуються попередньо навчені вектори ознак та комбіновані нейромережеві архітектури [3]-[5].

Процес розпізнавання зазвичай складається з кількох етапів. На початковій фазі застосовуються перетворення, які вирівнюють і нормалізують

текст, прибираючи нахили та викривлення. Далі відбувається вилучення ознак, де згорткові мережі формують карту характеристик, що описують почерк. Потім ці ознаки розглядаються як послідовність і обробляються рекурентними моделями, здатними враховувати контекст із обох боків. Завершальним етапом є прогнозування символів і формування слів [3]-[5].

Поряд із цим активно розвиваються методи, які будують розпізнавання навколо механізмів уваги. Такі архітектури дозволяють зосередитися на найбільш інформативних ділянках зображення чи послідовності, підвищуючи точність навіть для довгих текстів. Комбінація згорткових мереж, рекурентних блоків і уваги лежить в основі моделей типу seq2seq, де текст генерується символ за символом [3]-[5].

Окремий напрямок складає генерація синтетичних рукописних даних. Оскільки збирання великих анотованих корпусів є складним, застосовуються генеративно-змагальні мережі, здатні створювати нові зразки письма. Такі рішення не лише збагачують навчальні набори, а й дозволяють моделювати різні стилі почерку. Зокрема, деякі підходи пропонують метод частково контрольованого навчання, який поєднує марковані та немарковані приклади, підвищуючи адаптивність систем [4], [5].

У цілому розвиток методів розпізнавання рукописного тексту відображає тенденцію до об'єднання різних архітектур та використання синтетичних даних для подолання браку ресурсів. Поєднання згорткових та рекурентних нейронних мереж, трансформерів та генеративних підходів відкриває шлях до створення систем, здатних працювати з будь-якими стилями письма з високою точністю й ефективністю [4], [5].

Розпізнавання рукописного тексту є напрямом, що поєднує дослідження у сфері штучного інтелекту, машинного навчання та обробки зображень. Використання такого програмного забезпечення дозволяє автоматизувати перетворення рукописних символів у цифрову форму, що полегшує збереження та пошук інформації. Особливу цінність ця технологія

має в архівах, банківській сфері, освітніх закладах та медичних установах, де досі зберігається велика кількість рукописних документів [4], [5].

До переваг відносять зручність у роботі з великими обсягами даних, скорочення часу на обробку документів, зниження ризику людських помилок та можливість інтеграції в інші інформаційні системи. Однак існують і труднощі, серед яких варто відзначити високу залежність від якості рукопису, різноманіття індивідуальних почерків, наявності артефактів у відсканованих матеріалах і складність навчання моделей, здатних коректно працювати з неоднорідними зображеннями [4], [5].

Створення програмного забезпечення для розпізнавання рукописного тексту включає кілька послідовних етапів. Спочатку формується база зразків для тренування системи, після чого відбувається попередня обробка зображень, що може охоплювати фільтрацію шумів, бінаризацію та нормалізацію символів. Наступним кроком є застосування методів сегментації, які дозволяють виділити окремі слова або літери. На завершальному етапі моделі машинного навчання або нейронні мережі аналізують ці символи та перетворюють їх у текстовий формат. Подальше вдосконалення програмного забезпечення здійснюється шляхом навчання на нових наборах даних і використання адаптивних алгоритмів, здатних налаштовуватися під індивідуальні особливості почерку [5], [6].

Така технологія поступово стає невід'ємною частиною цифровізації, забезпечуючи більш ефективну роботу з документами та відкриваючи нові можливості для автоматизації процесів у різних сферах діяльності [5], [6].

Визначено, що розпізнавання рукописного тексту являє собою складну задачу, оскільки почерк людини відзначається великою варіативністю, нерівномірністю символів та їх накладанням. Для забезпечення точності програмні рішення в цій сфері використовують методи штучного інтелекту, зокрема нейронні мережі, здатні навчатися на великих масивах даних і виявляти закономірності навіть у складних почерках.

1.2 Аналіз програмного забезпечення для розпізнавання рукописного тексту

Проаналізуємо програмне забезпечення для розпізнавання рукописного тексту [7]-[13].

Програмне забезпечення Transkribus (рис. 1.1) являє собою комплексну платформу, що надає можливості обробки, аналізу та оцифрування письмових документів шляхом застосування штучного інтелекту. Система вирізняється здатністю ефективно працювати як з рукописними, так і з друкованими текстами, що особливо актуально для історичних матеріалів, наукових документів, архівів, бібліотек, освітніх і генеалогічних проєктів [8], [11].

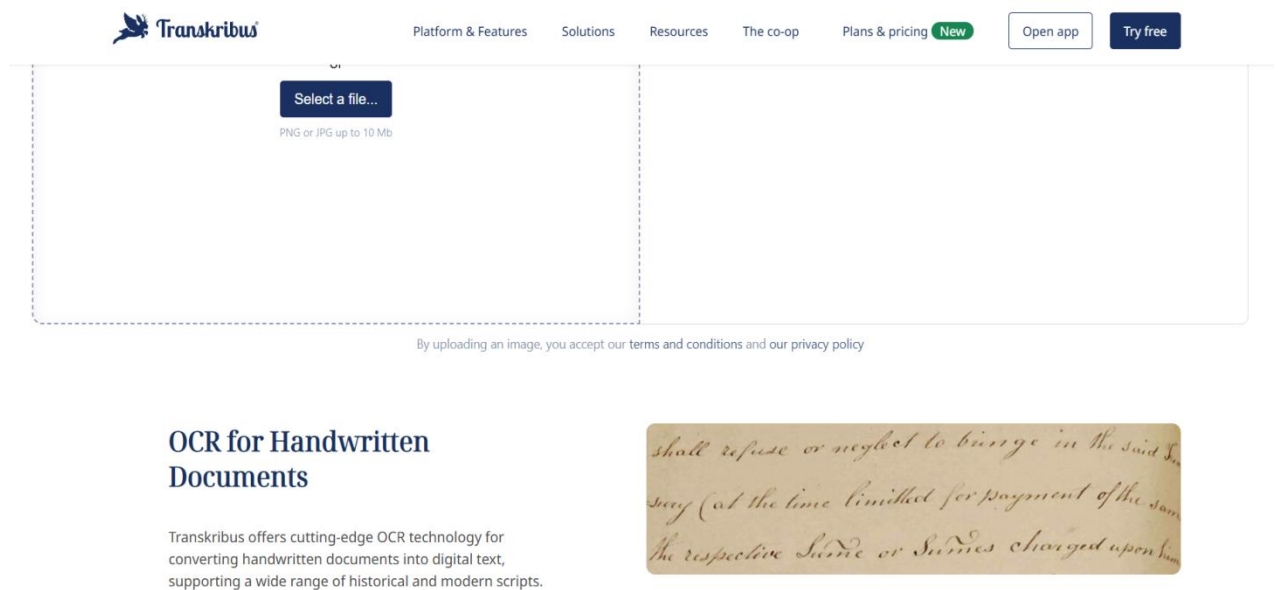


Рисунок 1.1 – Програмне забезпечення Transkribus [11]

Transkribus надає інструменти для навчання власних моделей розпізнавання тексту, що дозволяє адаптувати систему до характерних стилів письма конкретних документів або рукописів. Одночасно забезпечується підтримка співпраці через можливість налаштування тегів, метаданих, пошук по повному тексту та спільну роботу над колекціями [8], [11].

Завдяки вебінтерфейсу, програмне забезпечення Transkribus дозволяє редагувати розпізнаний текст без необхідності програмування, забезпечуючи легкий доступ до функцій як транскрипції, так і корекції результатів розпізнавання. Інфраструктура Transkribus охоплює архітектуру, побудовану на принципах глибокого навчання, яка здатна виявляти структуру документа – такі елементи, як заголовки, таблиці, текст, поля форм. Наявні публічні моделі охоплюють широкий діапазон мов, стилів письма та історичних періодів, а також є можливість створити і навчити власну модель для специфічного корпусу документів [8], [11].

Ключовим сценарієм використання Transkribus є надання доступу до оцифрованих і пошукових варіантів текстових матеріалів, що сприяє публікації, доступності та інтеграції результатів у потік робочих процесів дослідницьких та музейних організацій. Іншими словами, Transkribus перетворює рукописні та друковані джерела у зручний для пошуку, аналізу та поширення формат [8], [11].

Таким чином, Transkribus виступає універсальним інструментом автоматизованого розпізнавання текстів, що дозволяє об'єднувати машинне навчання, інтерактивну транскрипцію, роботу з метаданими, колаборацію та адаптивність у рамках єдиної платформи [8], [11].

Основними характеристиками програмного забезпечення Transkribus є такі [8], [11]:

- універсальність використання: програмне забезпечення Transkribus підтримує розпізнавання як рукописних, так і друкованих текстів у широкому діапазоні мов та стилів письма [8], [11];

- гнучке налаштування: програмне забезпечення Transkribus надає можливість навчання власних моделей розпізнавання, адаптованих під специфічні документи чи почерк [8], [11];

- підтримка колаборації: програмне забезпечення Transkribus забезпечує спільну роботу з колекціями, включно з додаванням тегів, метаданих та функцією пошуку [8], [11];

- інтерактивне редагування: програмне забезпечення Transkribus дозволяє виправляти результати розпізнавання без необхідності програмування через зручний інтерфейс [8], [11];
- структурний аналіз документів: програмне забезпечення Transkribus автоматично визначає заголовки, таблиці, абзаци та інші елементи сторінки та зображення [8], [11];
- доступність через Інтернет: програмне забезпечення Transkribus працює без інсталяції додаткового програмного забезпечення та доступна з будь-якого пристрою з інтернетом [8], [11];
- можливості публікації: програмне забезпечення Transkribus інтегрується з інструментами для створення онлайн архівів і надання доступу до оцифрованих матеріалів [8], [11];
- API-сумісність: програмне забезпечення Transkribus підтримує взаємодію з іншими програмними системами через інтерфейси прикладного програмування [8], [11].

Серед недоліків програмного забезпечення Transkribus можна відзначити обмеження у безкоштовному доступі до повного функціоналу, що зумовлює необхідність придбання додаткових пакетів для обробки значних обсягів даних. Також варто згадати залежність від стабільного інтернет-з'єднання, оскільки більшість операцій виконується у хмарному середовищі. Окремі користувачі можуть зіткнутися зі складнощами під час налаштування власних моделей розпізнавання через потребу в підготовці значних навчальних наборів. Крім того, точність роботи може знижуватися у випадку сильно пошкоджених або низькоякісних зображень, що вимагає додаткового етапу попередньої обробки матеріалів [8], [11].

Програма Mazes (рис. 1.2) являє собою програмне рішення, призначене для заміни традиційного способу введення тексту через клавіатуру інтуїтивним рукописним методом [9], [12].

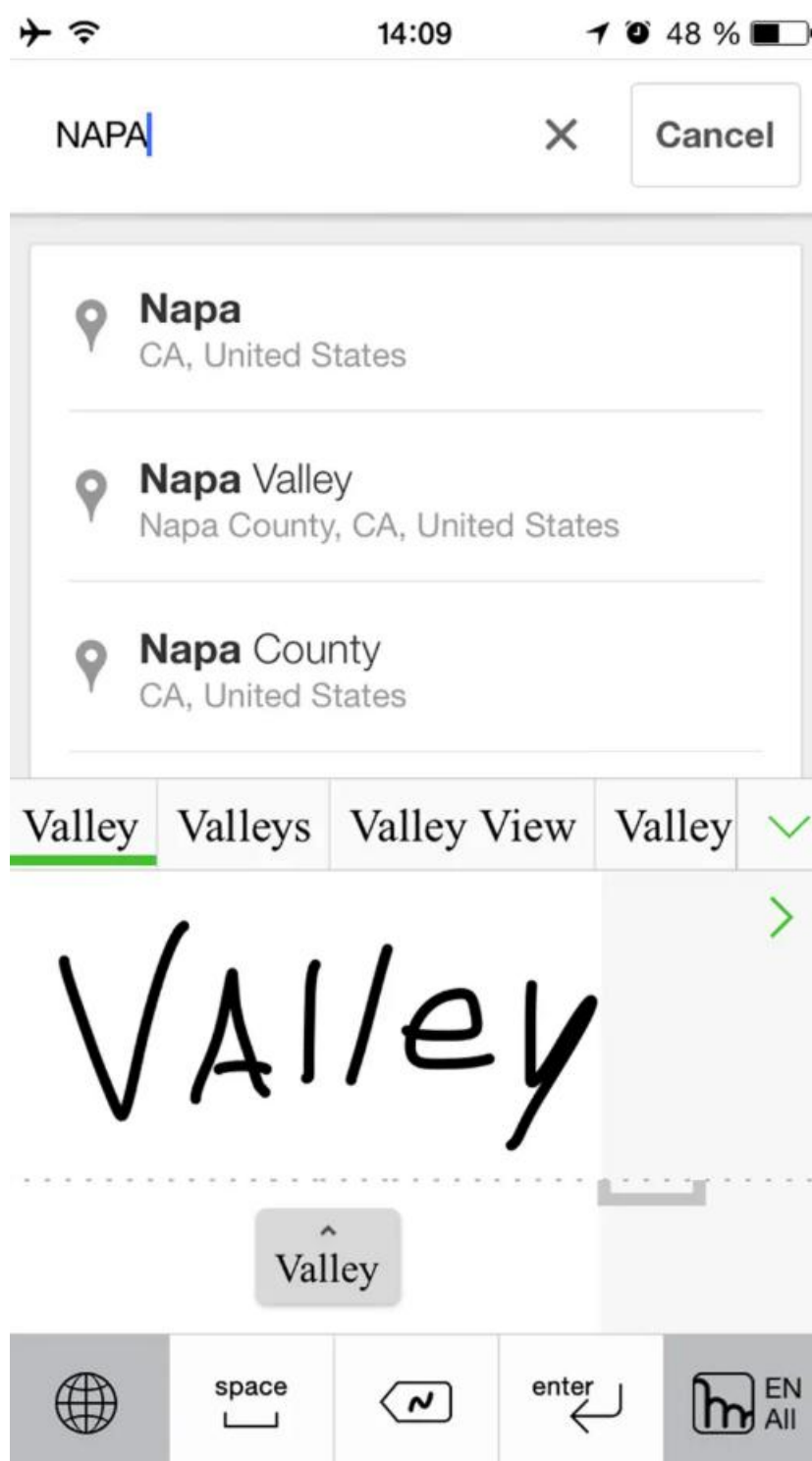


Рисунок 1.2 – Приклад роботи програмного забезпечення Mazes [12]

Програмне забезпечення Mazes здатне перетворювати введені від руки символи на друкований текст у динамічному режимі, забезпечуючи високу точність розпізнавання та можливість адаптації до індивідуального стилю письма. Передбачено використання механізмів прогнозування слів, що підвищує швидкість набору та зменшує кількість помилок, а також

налаштування словника шляхом додавання нових термінів і виразів, актуальних для користувача [9], [12].

Програмне забезпечення Mazes підтримує сумісність із широким спектром додатків, дозволяючи застосовувати рукописне введення для створення електронних листів, нотаток, публікацій у соціальних мережах, заповнення онлайн форм і пошуку в мережі. Програмне забезпечення підтримує багатомовність та додаткові мовні модулі, які можуть бути встановлені за потреби. Використання Mazes сприяє підвищенню продуктивності та комфорту взаємодії з цифровими пристроями, особливо у випадках, коли введення тексту від руки є більш природним або зручним, ніж використання клавіатури [9], [12].

Програмне забезпечення Mazes має такі особливості [9], [12]:

- рукописне введення в реальному часі: забезпечується миттєве перетворення символів, написаних від руки, у друкований текст із високим рівнем точності [9], [12];
- адаптивний словник: передбачена можливість автоматичного та ручного доповнення словника індивідуальними словами й виразами для підвищення персоналізації [9], [12];
- прогнозування тексту: реалізовано функцію передбачення наступних слів, що оптимізує швидкість набору та зменшує кількість помилок [9], [12];
- підтримка багатьох мов: доступна робота з різними мовами, зокрема шляхом встановлення додаткових мовних модулів [9], [12];
- сумісність із різними додатками: функціонал може використовуватися для введення тексту в будь-яких програмах пристрою, включаючи електронну пошту, соціальні мережі та браузері [9], [12];
- розширена взаємодія з інтерфейсом: забезпечується можливість швидкого заповнення вебформ, пошуку в Інтернеті та створення публікацій без потреби у використанні віртуальної клавіатури [9], [12].

Серед недоліків програмного забезпечення Mazes можна відзначити обмежену платформну сумісність, оскільки програма функціонує лише на

певних операційних системах, що обмежує коло користувачів. Крім того, для максимальної точності розпізнавання тексту часто потрібне звикання до стилю введення, що може створювати певні труднощі на початкових етапах використання. Варто також зазначити, що деякі функції, зокрема додаткові мовні модулі та розширені словники, доступні лише через додаткові покупки, що може ускладнити повне використання потенціалу програми [9], [12].

Програмне забезпечення Pen to Print (рис. 1.3) представляє собою рішення для оцифрування рукописних текстів, яке дозволяє перетворювати відскановані або сфотографовані документи на редагований цифровий формат. Система забезпечує витяг тексту з різноманітних паперових джерел, включно з особистими нотатками, листами, навчальними матеріалами та робочими списками, надаючи користувачам можливість зберігати отриманий текст на пристрої або в хмарних сервісах [9], [13].

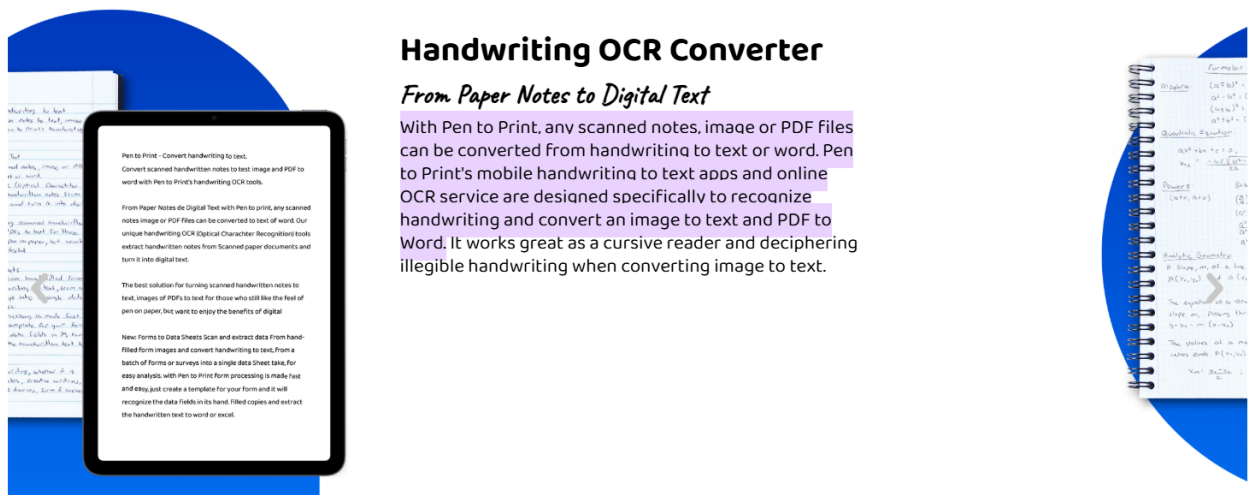


Рисунок 1.3 – Програмне забезпечення Pen to Print [13]

Програмне забезпечення Pen to Print підтримує різні стилі письма, зокрема друковані літери, скоропис і стандартні шрифти, що дозволяє точно відтворювати рукописну інформацію у цифровому вигляді. Програмне забезпечення Pen to Print надає користувачу зручні засоби редагування, пошуку та організації тексту, що робить його придатним для подальшого використання в робочих і навчальних процесах [9], [13].

Крім того, Pen to Print пропонує можливості інтеграції з популярними платформами обміну повідомленнями та текстовими процесорами, включаючи сервіси для створення нотаток та документи онлайн, що дозволяє ефективно використовувати оцифрований текст у різних цифрових середовищах. Програма підтримує декілька мов, що розширює її застосування серед користувачів із різних регіонів, і забезпечує швидкий та інтуїтивно зрозумілий процес конвертації рукописного тексту у зручний для редагування та поширення формат [9], [13].

Програмне забезпечення Pen to Print має такі особливості [9], [13]:

- оптичне розпізнавання тексту: програмне забезпечення Pen to Print витягує рукописні символи з відсканованих документів та перетворює їх на редагований цифровий текст [9], [13];
- підтримка різних стилів письма: програмне забезпечення Pen to Print обробляє друковані літери, скоропис та стандартні шрифти для точного відтворення тексту [9], [13];
- інтеграція з хмарними сервісами: програмне забезпечення Pen to Print дозволяє зберігати оцифрований текст у хмарних платформах для подальшого доступу та обміну [9], [13];
- сумісність із текстовими процесорами: програмне забезпечення Pen to Print забезпечує експорт у програми для редагування тексту, такі як Microsoft Word та Google Docs [9], [13];
- інтеграція з платформами обміну повідомленнями: програмне забезпечення Pen to Print надає можливість надсилати текст через сервіси на зразок WhatsApp, Messenger та Telegram [9], [13];
- багатомовна підтримка: програмне забезпечення Pen to Print працює з кількома мовами, що розширює коло користувачів і забезпечує універсальність застосування [9], [13];
- зручність редагування: програмне забезпечення Pen to Print надає інструменти для пошуку, копіювання та організації тексту безпосередньо у додатку [9], [13];

– адаптація до користувача: програмне забезпечення Pen to Print підтримує персоналізацію та налаштування текстових словників для підвищення точності розпізнавання [9], [13].

Серед недоліків програмного забезпечення Pen to Print можна відзначити обмежену точність розпізнавання тексту при обробці документів зі складним або малопомітним рукописним почерком, що вимагає додаткового редагування користувачем. Крім того, функціонал преміумальних підписок, який надає розширені можливості збереження та експорту тексту, доступний лише за плату, що може обмежувати повне використання програми. Також варто враховувати обмежену підтримку мов, що може ускладнювати роботу користувачів, які використовують тексти іншими мовами, ніж ті, що підтримуються додатком [9], [13].

Порівняльну характеристику програмного забезпечення для розпізнавання рукописного тексту наведено у таблиці 1.1.

Таблиця 1.1 – Порівняння програмного забезпечення для розпізнавання рукописного тексту

Критерій порівняння	Transkribus [11]	Mazec [12]	Pen to Print [13]
Висока точність розпізнавання	+	+—	+—
Багатомовність	+	+	+—
Можливість навчання власних моделей	+	+—	—
Зручність інтерфейсу	+—	+	+
Доступність на різних платформах	+—	+—	+—
Функціональність хмарного збереження	+	+—	+

За результатами проведеного аналізу можна зробити висновок, що у наш час існує досить багато програмних засобів для розпізнавання рукописного тексту. Проте деякі програмні засоби для розпізнавання рукописного тексту можуть використовувати технології штучного інтелекту та глибинного навчання, що є чутливими до нерозбірливого почерку, стилістичних відмінностей у написанні символів та індивідуальних особливостей письма.

Значний вплив на точність результатів справляють фактори, пов'язані з якістю вхідних даних, зокрема наявність шумів, викривлень або низька роздільна здатність зображення.

Певну проблему становить і коректне розпізнавання текстів, написаних у змішаних форматах, де використовуються різні мови, алфавіти або шрифти, що вимагає додаткових обчислювальних ресурсів та налаштувань моделі. Окремою проблемою є розпізнавання спеціалізованих позначень, формул чи скорочень, що часто зустрічаються у технічних, наукових або медичних записах. Застосування таких програмних рішень у реальному середовищі може бути обмежене високими вимогами до обчислювальної потужності, потребою у великих обсягах навчальних даних та необхідністю періодичного оновлення моделей.

Тому актуальною є розробка програмного забезпечення для розпізнавання рукописного тексту.

При розробці програмного забезпечення для розпізнавання рукописного тексту необхідно забезпечити такі функціональні вимоги:

- підтримка можливості завантаження зображень для подальшого розпізнавання тексту;
- підтримка можливості обрізання зображення користувачем для виділення області з текстом;
- можливість розпізнавання текстової інформації на зображенні;
- можливість збереження результату.

1.3 Висновки за розділом 1

Визначено, що розпізнавання рукописного тексту являє собою складну задачу, оскільки почерк людини відзначається великою варіативністю, нерівномірністю символів та їх накладанням. Для забезпечення точності програмні рішення в цій сфері використовують методи штучного інтелекту, зокрема нейронні мережі, здатні навчатися на великих масивах даних і виявляти закономірності навіть у складних почерках.

За результатами проведеного аналізу зроблено висновок, що у наш час існує досить багато програмних засобів для розпізнавання рукописного тексту. Проте деякі програмні засоби для розпізнавання рукописного тексту можуть використовувати технології штучного інтелекту та глибинного навчання, що є чутливими до нерозбірливого почерку, стилістичних відмінностей у написанні символів та індивідуальних особливостей письма. Значний вплив на точність результатів справляють фактори, пов'язані з якістю вхідних даних, зокрема наявність шумів, викривлень або низька роздільна здатність зображення. Певну проблему становить і коректне розпізнавання текстів, написаних у змішаних форматах, де використовуються різні мови, алфавіти або шрифти, що вимагає додаткових обчислювальних ресурсів та налаштувань моделі. Окремою проблемою є розпізнавання спеціалізованих позначень, формул чи скорочень, що часто зустрічаються у технічних, наукових або медичних записах. Застосування таких програмних рішень у реальному середовищі може бути обмежене високими вимогами до обчислювальної потужності, потребою у великих обсягах навчальних даних та необхідністю періодичного оновлення моделей. Тому актуальною є розробка програмного забезпечення для розпізнавання рукописного тексту.

Сформульовано функціональні вимоги до програмного забезпечення для розпізнавання рукописного тексту.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Вибір мови програмування

При розробці програмного забезпечення для розпізнавання рукописного тексту було використано мову програмування Python [14], [15].

Python відносять до універсальних мов високого рівня, виконання коду в якій здійснюється інтерпретатором, а динамічна типізація забезпечує значну свободу під час створення програм. Концепція побудови цієї мови дозволяє використовувати її як для розробки невеликих утиліт, так і для масштабних систем, що виконують ресурсоємні обчислення. Структура синтаксису відзначається простотою та логічністю, що позитивно впливає на зручність читання коду та полегшує його підтримку, особливо за умов колективної розробки та довгострокового використання програмних продуктів [14], [15].

Високий рівень популярності мови програмування Python пояснюється наявністю великої кількості бібліотек і фреймворків, які охоплюють широкий спектр завдань – від аналізу даних та впровадження алгоритмів штучного інтелекту до створення вебсервісів і автоматизації процесів. Сформована екосистема дає змогу поєднувати Python з іншими технологічними рішеннями та мовами, зберігаючи ефективність у випадках, коли першочергове значення має швидкість розробки, а не максимально можлива продуктивність виконання [14], [15].

Важливу роль відіграє глобальна спільнота, яка активно розвиває інструментарій, доповнює документацію та підтримує відкритий код, що забезпечує стабільну адаптацію мови до швидких змін у сфері інформаційних технологій [14], [15].

Завдяки універсальним можливостям, доступності синтаксису та розвиненій інфраструктурі мова програмування Python залишається дієвим засобом реалізації проєктів у науковій, інженерній, комерційній та освітній діяльності [14], [15].

Використання мови програмування Python для створення програмних систем, що виконують розпізнавання рукописного тексту, обґрунтовується поєднанням технічних, методологічних і практичних переваг. Основним чинником, що визначає доцільність використання мови програмування Python, є широка застосовність у сфері машинного навчання та обробки зображень, що надає можливість використовувати готові бібліотеки та фреймворки, такі як TensorFlow, PyTorch, Keras, OpenCV і Pillow. Інтеграція цих програмних інструментів дозволяє значно скоротити терміни розробки та підвищити ефективність реалізації алгоритмів аналізу графічної інформації та побудови моделей розпізнавання [14], [15].

Особливістю Python є його зрозуміла та логічна синтаксична структура, що знижує складність засвоєння мови та спрощує колективну роботу над проєктами, зосереджуючи увагу розробників на логіці алгоритмів, а не на деталях реалізації. Завдяки високому рівню абстракції створюються умови для швидкого прототипування, що є особливо актуальним на етапах досліджень і тестування моделей глибинного навчання [14], [15].

Додатковим чинником виступає активна підтримка спільноти розробників, яка регулярно поширює приклади коду, відкриті набори даних, навчальні матеріали та документацію. Це значно полегшує вирішення технічних завдань і сприяє інтеграції сучасних технологічних підходів у робочі рішення [14], [15].

Мультиплатформність мови програмування Python дозволяє створювати програми, сумісні з різними операційними системами, і забезпечує легку інтеграцію з іншими мовами та технологіями. Така універсальність сприяє масштабуванню програмних продуктів та адаптації їх до конкретних вимог користувачів [14], [15].

У підсумку, комбінація широкого набору інструментів, зрозумілого синтаксису, підтримки спільноти та високої гнучкості робить мову програмування Python оптимальним вибором для розробки програмних систем розпізнавання рукописного тексту [14], [15].

Обґрунтування вибору мови програмування для реалізації програмного забезпечення для розпізнавання рукописного тексту наведено у таблиці 2.1.

Таблиця 2.1 – Обґрунтування вибору мови програмування для реалізації програмного забезпечення для розпізнавання рукописного тексту

Критерій порівняння мов програмування	Мова програмування		
	Python	Java	C++
Бібліотеки для машинного навчання та комп'ютерного зору	+	+—	+—
Простота синтаксису та швидкість розробки	+	+—	—
Активність та розмір спільноти у сфері штучного інтелекту	+	+—	+—
Зручність для створення ПЗ для розпізнавання рукописного тексту	+	—	—
Гнучкість у прототипуванні та тестуванні моделей	+	+—	—
Ефективність для роботи з великими наборами даних	+	+—	+—

Отже, для реалізації програмного забезпечення для розпізнавання рукописного тексту обрано мову програмування Python через широке використання у сфері машинного навчання та комп'ютерного зору, що забезпечує розробникам доступ до великої кількості готових бібліотек та інструментів, застосування яких дає змогу значно скоротити час розробки та підвищити ефективність створення алгоритмів обробки зображень і побудови моделей розпізнавання. Крім того, Python вирізняється простою та зрозумілою синтаксичною структурою, що зменшує поріг входу для фахівців та полегшує

командну розробку, дозволяючи зосередитися на логіці алгоритмів, а не на складнощах реалізації. Високий рівень абстракції мови сприяє швидкому прототипуванню, що особливо важливо на етапі досліджень і експериментів із моделями глибинного навчання.

2.2 Вибір середовища розробки для створення програмного забезпечення для розпізнавання рукописного тексту

В якості середовища розробки для створення програмного забезпечення для розпізнавання рукописного тексту було обрано середовище Visual Studio Code [16], [17].

Середовище розробки Visual Studio Code дозволяє максимально налаштовуватися під конкретні потреби розробника ПЗ. Його конструкція забезпечує можливість розширення базового функціоналу через підключення додаткових модулів, що охоплюють як спеціалізовані інструменти, так і засоби інтеграції з хмарними сервісами та зовнішніми системами. Гнучкі налаштування інтерфейсу створюють умови для організації робочого простору відповідно до особливостей проєкту, що підвищує продуктивність роботи та комфорт виконання завдань [16], [17].

Середовище Visual Studio Code підтримує актуальні стандарти програмування і забезпечує стабільну роботу з різними мовами, навіть при обробці великих обсягів коду. Особлива увага приділяється взаємодії з програмними інструментами налагодження, тестування та автоматизації, що робить його придатним як для швидкого створення прототипів, так і для довготривалого промислового застосування. Постійне оновлення функціоналу Visual Studio Code та активна підтримка спільноти користувачів і розробників дозволяють середовищу залишатися сучасним і відповідати новітнім технологічним тенденціям [16], [17].

Поєднання модульності, високої продуктивності та можливості інтеграції нових технологій визначає Visual Studio Code як універсальний

інструмент, який ефективно використовується для широкого спектра завдань, від навчальних проєктів до складних комерційних рішень [16], [17].

Зручний інтерфейс та інтелектуальні підказки коду сприяють підвищенню швидкості написання програм, зменшують ймовірність синтаксичних помилок і полегшують навігацію у великих проєктах [16], [17].

Використання середовища Visual Studio Code для розробки програмних систем, здатних розпізнавати рукописний текст, обґрунтовується поєднанням багатofункціональності, комфортності роботи та широких можливостей налаштування. Це середовище підтримує численні мови програмування, зокрема Python, який широко застосовується у галузях штучного інтелекту та обробки зображень. Завдяки розширенням можна легко інтегрувати бібліотеки для машинного навчання, системи контролю версій, інструменти налагодження та автоматизації процесів без необхідності змінювати робоче середовище [16], [17].

Інтерфейс з інтелектуальними підказками коду сприяє прискоренню розробки, зменшенню кількості синтаксичних помилок та полегшенню орієнтування у великих проєктах. Вбудовані можливості роботи з системами контролю версій забезпечують ефективну організацію командної розробки, що є важливим для складних проєктів із потребою регулярного оновлення та тестування [16], [17].

Мультиплатформність Visual Studio Code дозволяє використовувати його на Windows, macOS та Linux без втрати функціональності, а відносно невисокі вимоги до апаратних ресурсів роблять його доступним для широкого кола розробників [16], [17].

У підсумку, завдяки розширюваності, підтримці сучасних технологій, зручності роботи з Python та оптимізації робочих процесів, Visual Studio Code виступає ефективним інструментом для створення програмного забезпечення, що реалізує розпізнавання рукописного тексту [16], [17].

Обґрунтування вибору середовища розробки для створення програмного забезпечення для розпізнавання рукописного тексту наведено у таблиці 2.2.

Таблиця 2.2 – Обґрунтування вибору середовища розробки для створення програмного забезпечення для розпізнавання рукописного тексту

Критерій порівняння середовищ розробки	Середовища розробки		
	Visual Studio Code	PyCharm	Jupyter Notebook
Легкість налаштування та встановлення	+	+—	+
Гнучкість та розширюваність за рахунок плагінів	+	+—	—
Зручність роботи з великими проєктами	+	+	—
Підтримка інструментів машинного навчання та ШІ	+	+	+—
Зручність налагодження коду	+	+	—

Таким чином, для створення програмного забезпечення для розпізнавання рукописного тексту обрано середовище розробки Visual Studio Code завдяки поєднанню функціональності, зручності використання та гнучкості налаштувань. Завдяки розширенням розробник може інтегрувати бібліотеки для машинного навчання, системи керування версіями, інструменти для налагодження та автоматизації робочих процесів без потреби у зміні середовища. Крім того, зручний інтерфейс та інтелектуальні підказки коду

сприяють підвищенню швидкості написання програм, зменшують ймовірність синтаксичних помилок і полегшують навігацію у великих проєктах.

2.3 Розробка моделі взаємодії користувача з програмним забезпеченням для розпізнавання рукописного тексту

З метою можливості розробки програмного забезпечення для розпізнавання рукописного тексту створимо відповідну модель у вигляді діаграми прецедентів.

При розробленні моделі взаємодії користувача з програмним забезпеченням для розпізнавання рукописного тексту враховуємо те, що користувачі програми мають можливості завантаження зображень, отримання результату обробки зображень, редагування отриманого результату, збереження результату у файл.

Основний успішний сценарій користувача при роботі з програмним забезпеченням для розпізнавання рукописного тексту можна описати таким чином. У межах функціонування програмного забезпечення для розпізнавання рукописного тексту взаємодія з користувачем передбачає кілька послідовних етапів. Спочатку здійснюється завантаження зображень, що слугують джерелом даних для подальшої обробки. Для цього використовується інтерфейс, який дозволяє обрати вільну позицію з доступних і додати файл формату з графічним зображенням. Перед початком роботи система перевіряє коректність формату та допустимий розмір, а у випадку невідповідності виводить повідомлення про помилку. Після успішного додавання виконується автоматичний перехід до етапу вибору області з текстом. Далі відбувається виділення текстових зон на зображенні з можливістю збереження отриманого фрагмента. Якщо весь вміст зображення містить текст, передбачено обрізання без попереднього виділення. Результатом цього кроку є збереження скоригованої фотографії в окремому каталозі, що підготовлює її до подальшої обробки системою. Після виконання алгоритмів розпізнавання у відведеній

області інтерфейсу з'являється текстовий результат, який може бути змінений для усунення похибок розпізнавання. Редагування здійснюється безпосередньо у текстовому полі з урахуванням обмеження на максимальну кількість символів. Заключним етапом є збереження підсумкового тексту в окремий файл. Для цього передбачено натискання відповідної кнопки, після чого результат записується у текстовий документ. Якщо виявляються недопустимі символи, система повідомляє про помилку та пропонує виправлення. У підсумку створюється повний цикл роботи – від завантаження вхідних даних до формування готового текстового файлу.

Враховуючи сценарії роботи користувача програмного забезпечення для розпізнавання рукописного тексту при роботі з програмою, модель взаємодії користувача з програмним забезпеченням для розпізнавання рукописного тексту подамо у вигляді діаграми прецедентів, зображеної на рис. 2.1.

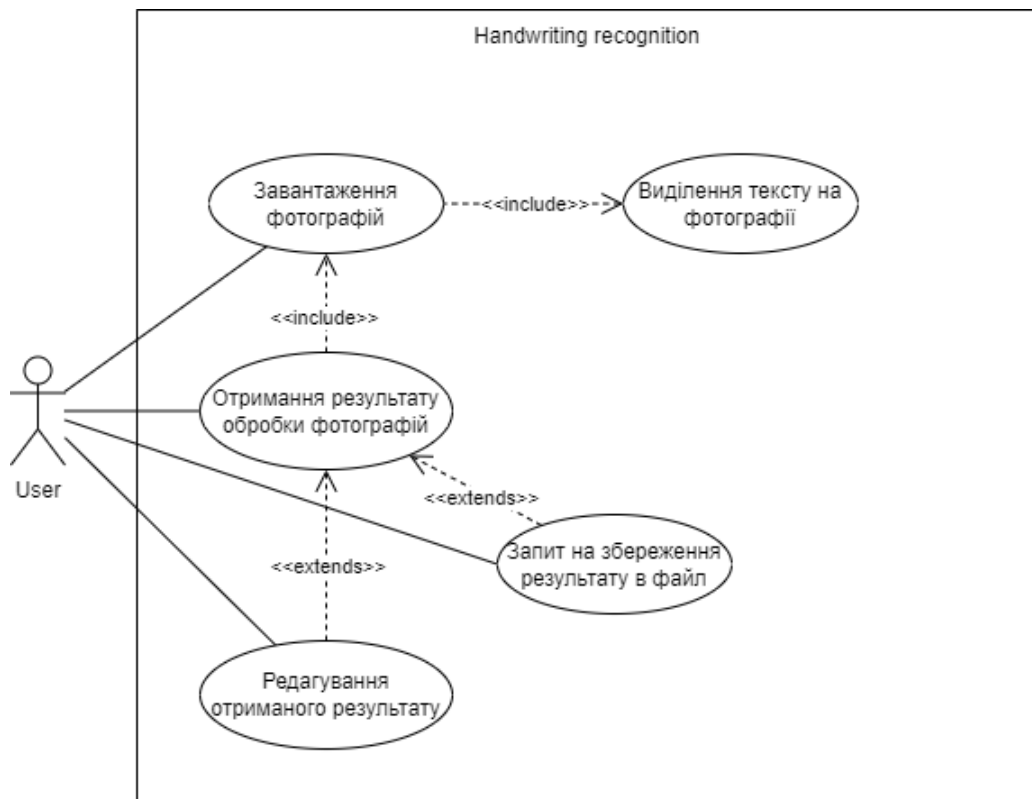


Рисунок 2.1 – Модель взаємодії користувача з програмним забезпеченням для розпізнавання рукописного тексту

Таким чином, створено модель взаємодії користувача з програмним забезпеченням для розпізнавання рукописного тексту, яка подана у вигляді діаграми прецедентів, та дозволяє розроблювати програмне забезпечення для розпізнавання рукописного тексту.

2.4 Висновки за розділом 2

Для реалізації програмного забезпечення для розпізнавання рукописного тексту обрано мову програмування Python через широке використання у сфері машинного навчання та комп'ютерного зору, що забезпечує розробникам доступ до великої кількості готових бібліотек та інструментів, застосування яких дає змогу значно скоротити час розробки та підвищити ефективність створення алгоритмів обробки зображень і побудови моделей розпізнавання.

Для створення програмного забезпечення для розпізнавання рукописного тексту обрано середовище розробки Visual Studio Code завдяки поєднанню функціональності, зручності використання та гнучкості налаштувань. Завдяки розширенням розробник може інтегрувати бібліотеки для машинного навчання, системи керування версіями, інструменти для налагодження та автоматизації робочих процесів без потреби у зміні середовища.

Створено модель взаємодії користувача з програмним забезпеченням для розпізнавання рукописного тексту, яка подана у вигляді діаграми прецедентів, та дозволяє розроблювати програмне забезпечення для розпізнавання рукописного тексту.

3 ОПИС ПРОГРАМИ

3.1 Структура програмного забезпечення для розпізнавання рукописного тексту

Структуру програмного забезпечення для розпізнавання рукописного тексту наведено у вигляді діаграми класів на рис. 3.1.

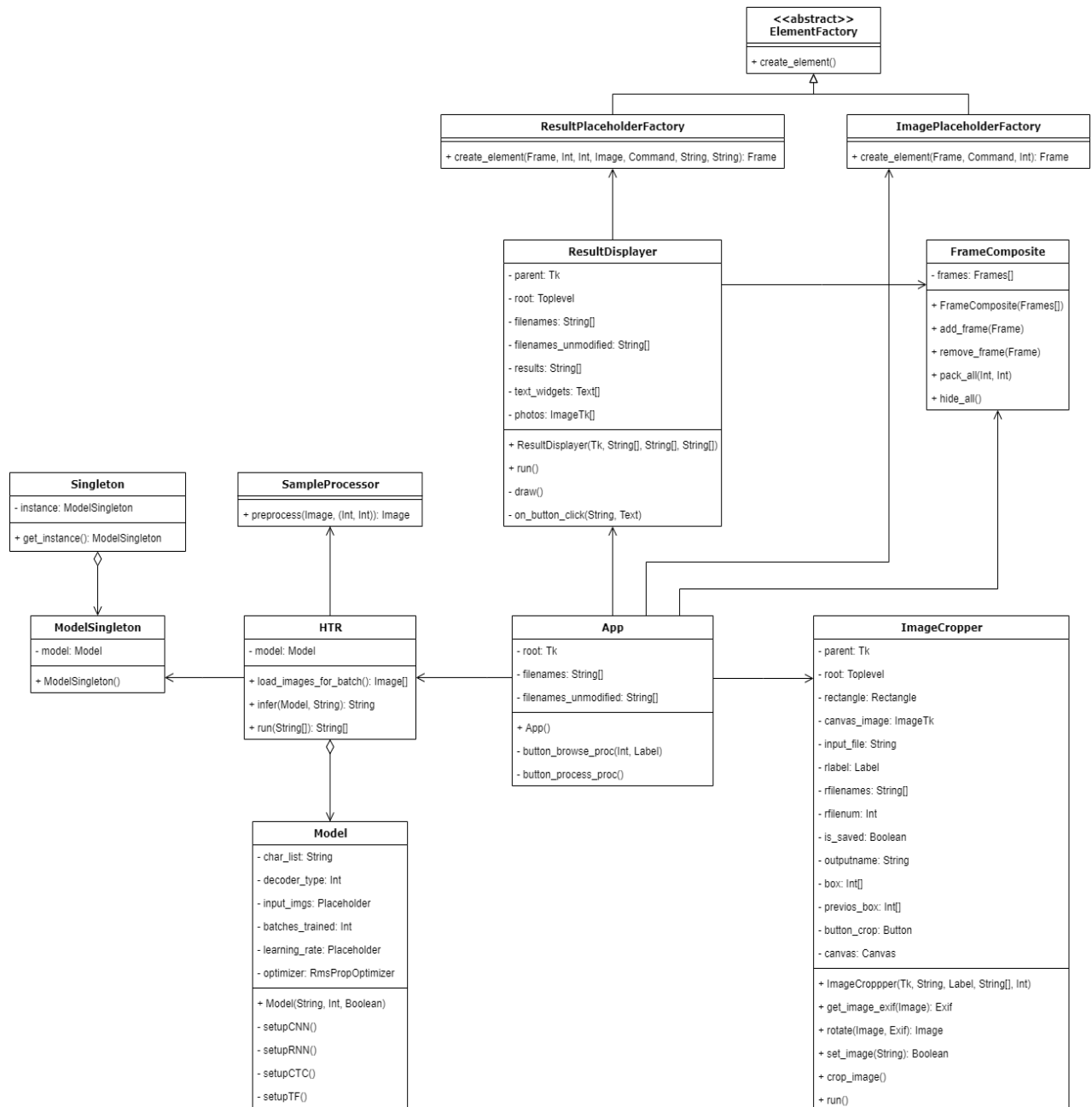


Рисунок 3.1 – Структура програмного забезпечення для розпізнавання рукописного тексту у вигляді діаграми класів

Діаграма класів програмного забезпечення для розпізнавання рукописного тексту відображає взаємозв'язки між основними компонентами, що забезпечують роботу системи.

Центральним елементом є клас App, який виступає керуючим центром усіх процесів та ініціює роботу інших об'єктів. Його взаємозв'язки з HTR і ImageCropper реалізовані у вигляді асоціацій, що забезпечують виклик відповідних методів під час завантаження, обробки та підготовки зображень до розпізнавання. Клас ImageCropper відповідає за виконання операцій виділення області з текстом і передає результат до SampleProcessor, який формує дані у форматі, сумісному з моделлю розпізнавання.

Процес розпізнавання реалізується через клас HTR, який взаємодіє з класом Model для отримання прогнозу на основі навченої нейронної мережі. Доступ до екземпляра моделі здійснюється за допомогою класу ModelSingleton, що забезпечує єдину точку створення та використання моделі в межах усієї програми.

Для обробки вхідних зразків передбачено клас SampleProcessor, який готує зображення до передачі в модель, виконуючи нормалізацію та інші необхідні трансформації. Результати роботи системи виводяться через клас ResultDisplayer, який відповідає за візуалізацію розпізнаного тексту в інтерфейсі користувача.

Структурна організація інтерфейсних елементів описана класом FrameComposite, що дозволяє комбінувати окремі віконні елементи в єдину композицію. Сукупність цих класів утворює цілісну архітектуру, в якій кожен компонент має чітко визначену роль, а взаємодія між ними забезпечує повний цикл обробки – від завантаження та підготовки зображення до відображення кінцевого результату.

Отже, структура програмного забезпечення для розпізнавання рукописного тексту побудована з урахуванням модульності, що робить можливим додавання нових алгоритмів або розширення функціоналу без зміни основних компонентів. Кожен клас має чітко визначені методи та

атрибути, при цьому публічні інтерфейси відповідають за зовнішню взаємодію, а приватні – за внутрішню логіку. Між класами простежуються як асоціативні, так і композиційні зв'язки, що відображає різний ступінь залежності між об'єктами. Структура діаграми класів узгоджується з іншими діаграмами, оскільки всі класи реалізують кроки сценаріїв використання, починаючи від завантаження зображення й закінчуючи редагуванням та збереженням результатів. Такий підхід забезпечує масштабованість і підтримуваність програмної системи, а також дозволяє інтегрувати нові технології розпізнавання без порушення існуючої логіки роботи.

3.2 Функціонування програмного забезпечення для розпізнавання рукописного тексту

Функціонування програмного забезпечення для розпізнавання рукописного тексту подамо за допомогою діаграм (рис. 3.2-3.4), що описують процеси функціонування системи на певних етапах.

На рис. 3.2 наведено схему функціонування програмного забезпечення при завантаженні зображення користувачем.

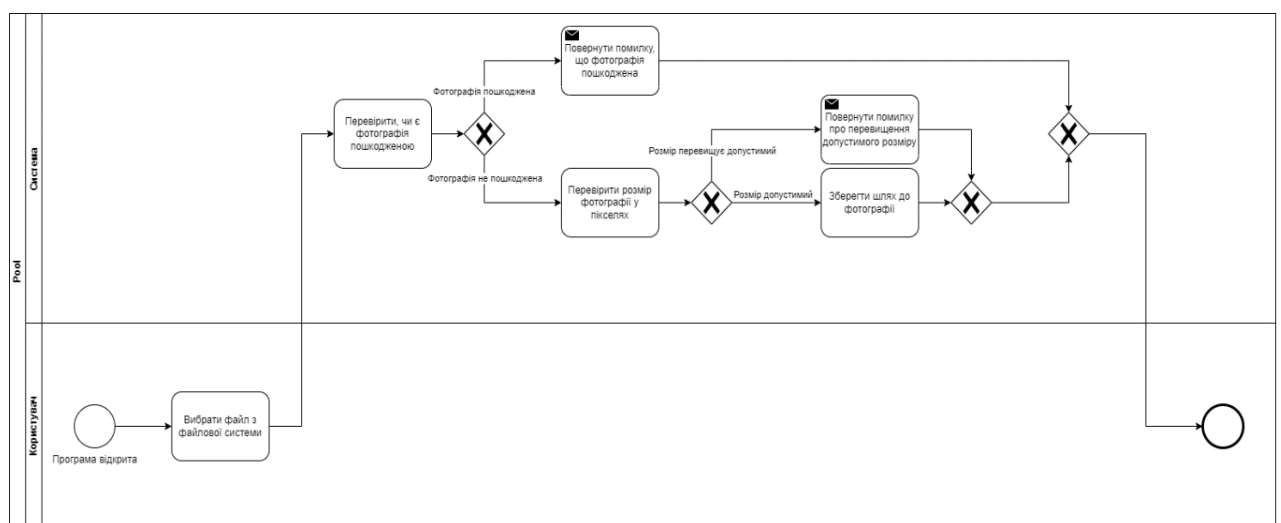


Рисунок 3.2 – Схема функціонування програмного забезпечення при завантаженні зображення користувачем

Процес завантаження зображення користувачем починається з ініціювання виклику інструменту вибору файлів, після чого з локального сховища обирається необхідне зображення та підтверджується його використання. Далі відбувається автоматична перевірка цілісності файлу, і у випадку виявлення пошкоджень формується повідомлення про помилку. Наступним етапом здійснюється контроль параметрів зображення, зокрема висоти та ширини, із фіксацією перевищення допустимих значень і виведенням відповідного попередження у разі порушень. Після успішного проходження перевірок вибраний файл переноситься до тимчасового буфера для подальшої обробки.

На рис. 3.3 наведено схему функціонування програмного забезпечення при обрізанні зображення користувачем.

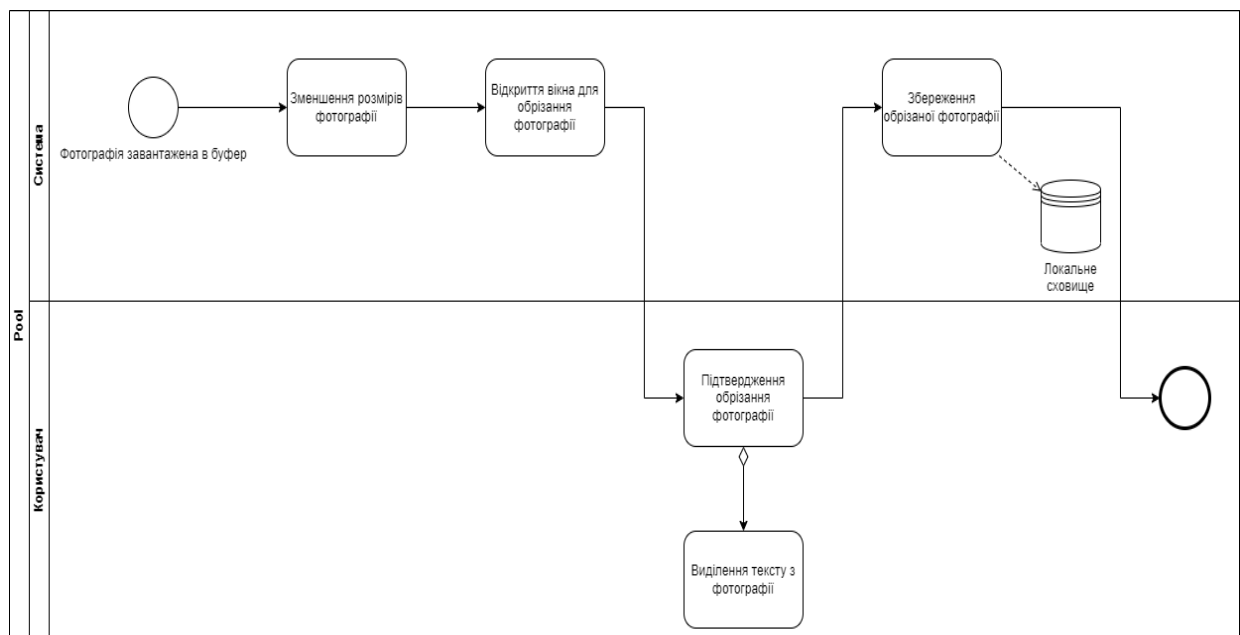


Рисунок 3.3 – Схема функціонування програмного забезпечення при обрізанні зображення користувачем

Після завантаження зображення відкривається окреме вікно перегляду, у якому відображається вибрана фотографія. У межах цього вікна виконується позначення ділянки, що містить потрібний текст, з подальшим підтвердженням дії. Після завершення вибору програмне забезпечення для

розпізнавання рукописного тексту здійснює обрізання зображення згідно з заданими межами та розміщує отриманий фрагмент у тимчасовому сховищі для подальшої обробки.

На рис. 3.4 наведено схему функціонування програмного забезпечення при зберіганні результатів користувачем.

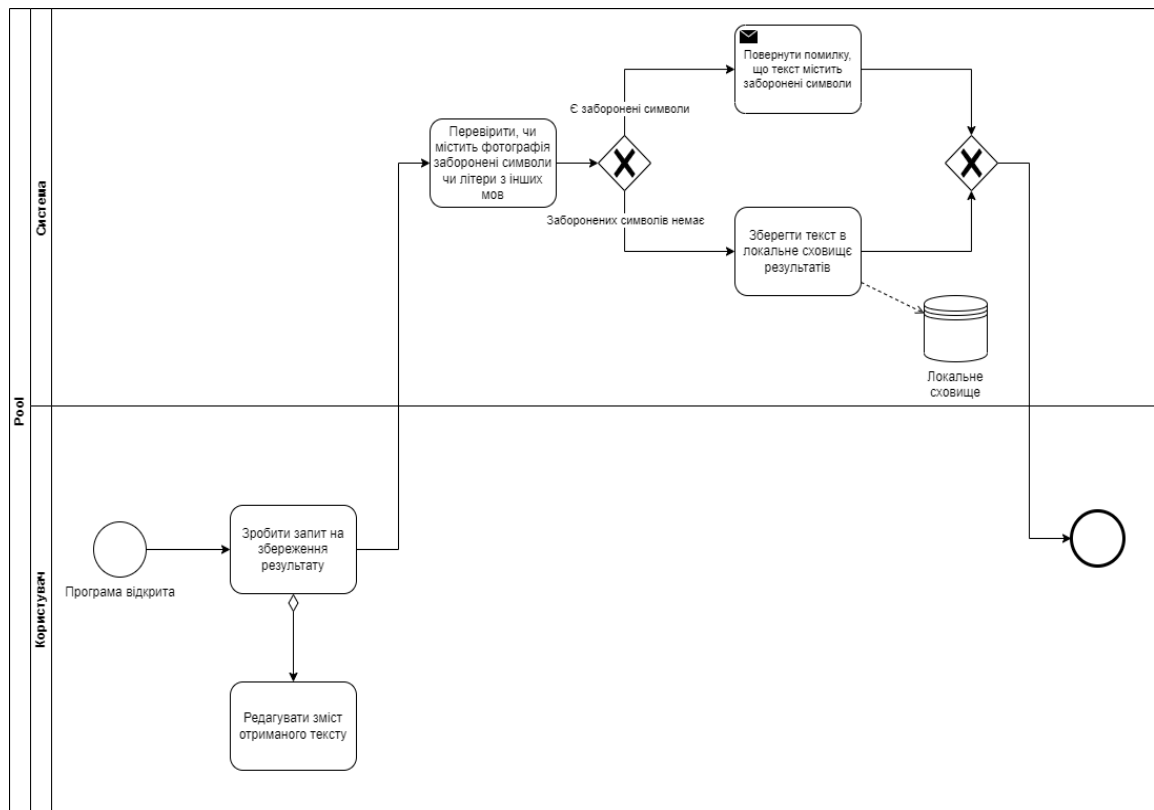


Рисунок 3.4 – Схема функціонування програмного забезпечення при зберіганні результатів користувачем

Під час збереження результатів редагований текст, отриманий після розпізнавання, передається системі із запитом на створення окремого файлу. Перед записом здійснюється автоматична перевірка вмісту на наявність недопустимих символів та знаків, що не належать до англійської абетки. Якщо такі елементи виявляються, формується повідомлення про помилку, і процес збереження призупиняється. У разі відсутності порушень текстовий результат успішно записується до окремого файлу у визначеній папці з результатами.

3.3 Синтез нейромережевої моделі для розпізнавання рукописного тексту

У програмному забезпеченні реалізовано нейромережеву архітектуру, що поєднує в собі кілька взаємодоповнюючих етапів обробки вхідних зображень, забезпечуючи високу точність розпізнавання рукописного тексту.

Графічне подання нейромережевої моделі для розпізнавання рукописного тексту наведено на рис. 3.5.

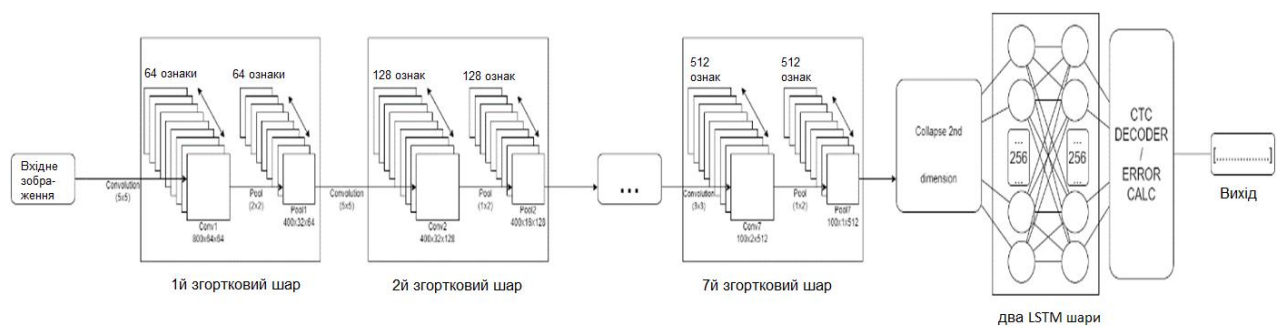


Рисунок 3.5 – Графічне подання нейромережевої моделі для розпізнавання рукописного тексту

На початковому етапі модель застосовує послідовність семи згорткових шарів, кожен з яких формує набір ознак, вилучаючи зображальні патерни різного рівня складності – від базових контурів і штрихів до більш складних структур, що відповідають почерку людини. Згорткові шари поступово зменшують просторову розмірність, водночас підвищуючи щільність та інформативність ознакового простору, що дозволяє зосередитися на ключових фрагментах, важливих для ідентифікації символів.

Далі отримані ознаки передаються до рекурентного модуля, який складається з двох двонаправлених LSTM-шарів. Цей етап відповідає за аналіз послідовних залежностей між символами та контекстних зв'язків у межах рядка тексту. Використання LSTM забезпечує здатність моделі враховувати як

попередні, так і наступні символи при розпізнаванні, що особливо важливо у випадках, коли почерк є нерівномірним або має неоднозначні елементи.

Заключним етапом є застосування шару CTC (Connectionist Temporal Classification), який виконує перетворення вихідних даних рекурентних шарів у остаточний текстовий результат. Цей механізм дозволяє синхронізувати послідовність виходів моделі з правильною послідовністю символів, не потребуючи попередньої жорсткої сегментації зображення на окремі букви. CTC-алгоритм ефективно справляється з варіативною довжиною текстових фрагментів і дозволяє моделі працювати навіть із неповністю вирівняними даними.

Уся структура побудована таким чином, щоб поєднувати потужність згорткових мереж у виявленні візуальних ознак та переваги рекурентних мереж у розумінні контексту. Така комбінація, доповнена механізмом CTC, робить модель здатною працювати з широким спектром почерків, різними умовами зйомки та нерівномірними пропорціями тексту, зберігаючи високу швидкість обробки та точність результатів.

Для підготовки нейромережевої моделі до розпізнавання рукописного тексту застосовувався набір даних IAM word handwriting [18], який містить велику кількість зображень, що представляють різні варіації почерків, стилів написання та умов сканування. Кожен елемент набору даних (рис. 3.6) супроводжується метаданими, у яких зазначено фізичні параметри зображення, інформацію про його стан, а також текстовий вміст, що слугує цільовим значенням під час навчання. Завдяки цим супровідним даним зображення проходять автоматизовану перевірку, що дозволяє відфільтрувати пошкоджені або некоректні зразки, а також адаптувати їх розмір і формат для подачі у нейронну мережу.



Рисунок 3.6 – Набір даних для синтезу неймережевої моделі для розпізнавання рукописного тексту

Перед початком основного навчання неймережевої моделі зображення нормалізуються за розміром, коригуються контраст та яскравість, а також виконується бінаризація для зменшення впливу шумів. Після цього вони подаються на вхід згорткових шарів, які вилучають ключові візуальні ознаки, що далі передаються рекурентним модулям для аналізу послідовностей. Під час тренування оптимізація ваг виконується із використанням алгоритму зворотного поширення похибки, адаптованого для роботи з СТС-функцією втрат, яка коректно обробляє невіривняні послідовності.

У ході навчання модель поступово вдосконалює здатність узгоджувати вхідні зображення з відповідними текстовими мітками, зменшуючи кількість помилок розпізнавання та покращуючи стабільність результатів навіть на почерках, які значно відрізняються від тих, що зустрічалися у тренувальному наборі. Такий підхід забезпечує не лише високу точність неймережевої моделі для розпізнавання рукописного тексту у стандартних випадках, але й

стійкість до нестандартних стилів письма та варіацій у якості вхідних зображень.

3.4 Проєктування інтерфейсу програмного забезпечення для розпізнавання рукописного тексту

Проєктування інтерфейсу взаємодії користувача з програмним забезпеченням для розпізнавання рукописного тексту виконано з урахуванням вимог технічного завдання. При розробленні програмного забезпечення для розпізнавання рукописного тексту враховані функціональні вимоги до програми:

- підтримка можливості завантаження зображень для подальшого розпізнавання тексту;
- підтримка можливості обрізання зображення користувачем для виділення області з текстом;
- можливість розпізнавання текстової інформації на зображенні;
- можливість збереження результату.

Інтерфейс головної форми програмного забезпечення для розпізнавання рукописного тексту наведено на рис. 3.7.

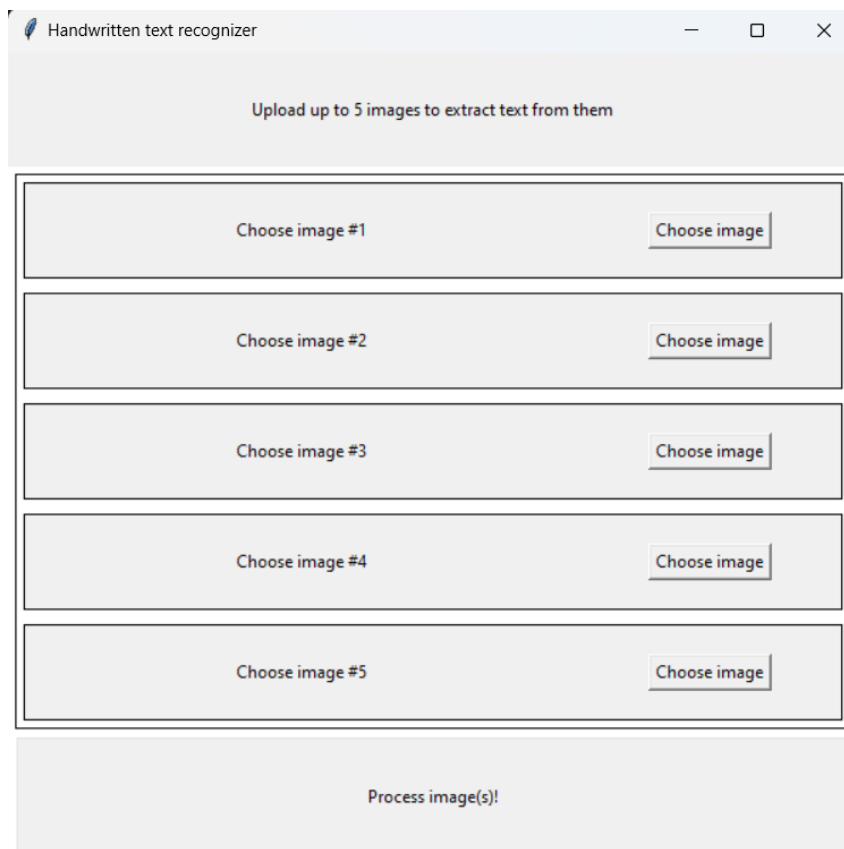


Рисунок 3.7 – Інтерфейс головної форми програмного забезпечення для розпізнавання рукописного тексту

Форма виділення області з текстом на зображенні наведена на рис. 3.8.

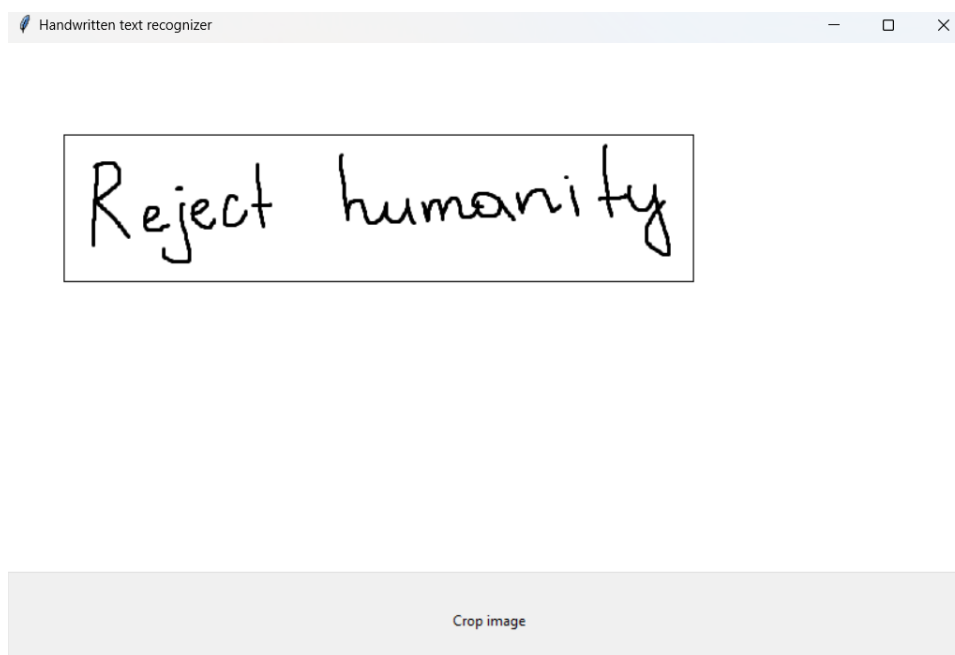


Рисунок 3.8 – Виділення області з текстом на зображенні

Форма виведення результату розпізнавання рукописного тексту наведена на рис. 3.9.

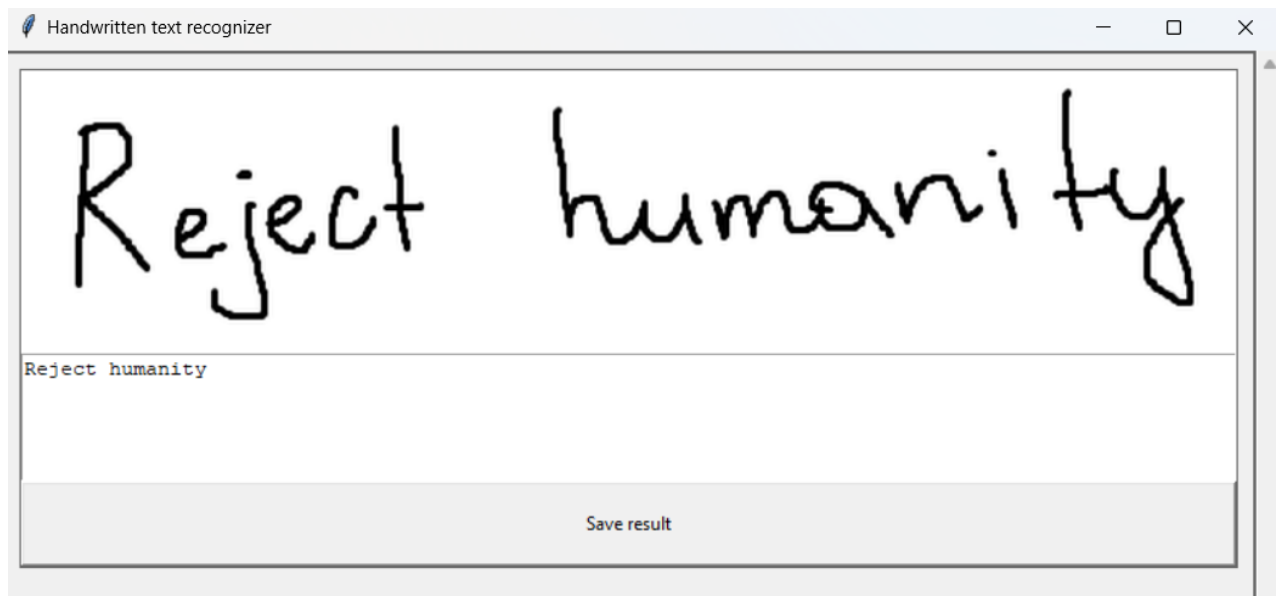


Рисунок 3.9 – Форма виведення результату розпізнавання рукописного тексту

3.5 Висновки за розділом 3

Розроблено програмне забезпечення для розпізнавання рукописного тексту.

Запропоновано структуру програмного забезпечення для розпізнавання рукописного тексту, що побудована з урахуванням модульності, що робить можливим додавання нових алгоритмів або розширення функціоналу без зміни основних компонентів. Визначено, що основними класами програмного забезпечення для розпізнавання рукописного тексту є: App, ImageCropper, SampleProcessor, ResultDisplayer, FrameComposite та ін.

Функціонування програмного забезпечення для розпізнавання рукописного тексту подано за допомогою діаграм, що описують процеси функціонування системи на певних етапах.

Визначено, що у програмному забезпеченні реалізовано нейромережеву архітектуру, що поєднує в собі кілька взаємодоповнюючих етапів обробки

вхідних зображень, забезпечуючи високу точність розпізнавання рукописного тексту. Структура нейромережі побудована таким чином, щоб поєднувати потужність згорткових мереж у виявленні візуальних ознак та переваги рекурентних мереж у розумінні контексту. Така комбінація робить модель здатною працювати з широким спектром почерків, різними умовами зйомки та нерівномірними пропорціями тексту, зберігаючи високу швидкість обробки та точність результатів.

Виконано проєктування інтерфейсу взаємодії користувача з програмним забезпеченням для розпізнавання рукописного тексту.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення й умови застосування програми

Програма призначена для розпізнавання рукописного тексту. Програма написана на мові Python через широке використання у сфері машинного навчання та комп'ютерного зору, що забезпечує розробникам доступ до великої кількості готових бібліотек та інструментів, застосування яких дає змогу значно скоротити час розробки та підвищити ефективність створення алгоритмів обробки зображень і побудови моделей розпізнавання. В якості середовища розробки для створення програмного забезпечення для розпізнавання рукописного тексту обрано Visual Studio Code завдяки поєднанню функціональності, зручності використання та гнучкості налаштувань.

Програмне забезпечення для розпізнавання рукописного тексту забезпечує виконання таких функцій:

- підтримка можливості завантаження зображень для подальшого розпізнавання тексту;
- підтримка можливості обрізання зображення користувачем для виділення області з текстом;
- можливість розпізнавання текстової інформації на зображенні;
- можливість збереження результату.

Для роботи програмного забезпечення для розпізнавання рукописного тексту потрібні такі технічні та програмні ресурси: для обчислювальної частини бажано наявність процесора з тактовою частотою не нижче 4 ГГц, що забезпечує ефективну обробку великих обсягів даних, а також багатоядерної архітектури для підтримки паралельних обчислень. Обсяг оперативної пам'яті має бути достатнім для завантаження моделей машинного навчання та обробки зображень, при цьому оптимально не менше восьми гігабайт, тоді як для високопродуктивних задач, пов'язаних із великими наборами даних,

доцільне використання значень від шістнадцяти гігабайт і більше. Для зберігання даних рекомендується використовувати накопичувачі достатнього обсягу (від 256 гігабайт), що забезпечують швидкий доступ до навчальних і тестових наборів даних.

Щодо програмного забезпечення, необхідно мати операційну систему та інтерпретатор Python. Крім того, середовище розробки має забезпечувати налагодження коду та можливість розширення функціоналу шляхом підключення додаткових модулів.

Функціональні характеристики розробленого програмного забезпечення для розпізнавання рукописного тексту наведено у технічному завданні (додаток А). Фрагмент тексту програмного забезпечення для розпізнавання рукописного тексту наведено у додатку Б.

4.2 Характеристики програми для розпізнавання рукописного тексту

Програмне забезпечення для розпізнавання рукописного тексту характеризується здатністю автоматично перетворювати зображення з рукописним письмом на цифровий формат тексту, що забезпечує подальшу обробку та зберігання даних у комп'ютерних системах. Воно забезпечує прийнятну точність розпізнавання різних стилів письма, та здатне адаптуватися до варіативності символів і форматів тексту.

До особливостей такого програмного забезпечення належить інтеграція алгоритмів машинного навчання та нейромережевих технологій, що дозволяє підвищувати якість розпізнавання навіть у складних умовах, коли текст містить нерозбірливі або частково пошкоджені символи. Програми відрізняються здатністю працювати з різними форматами файлів із зображеннями, що забезпечує універсальність у застосуванні.

У сукупності зазначені характеристики забезпечують ефективність, надійність та адаптивність програмного забезпечення для розпізнавання

рукописного тексту, що робить його придатним для використання при розв'язанні завдань різного призначення.

4.3 Інструкція по експлуатації програми

4.3.1 Звернення до програми

Для запуску програмного забезпечення для розпізнавання рукописного тексту необхідно запустити відповідний ру-файл.

4.3.2 Вхідні й вихідні дані

Вхідними даними для програмного забезпечення з розпізнавання рукописного тексту виступають зображення рукописного письма, отримані зі сканованих документів, фотографій або цифрових нотаток. Ці дані можуть мати різну якість, формат і роздільну здатність, що впливає на процес обробки та точність розпізнавання.

Вихідними даними програмного забезпечення для розпізнавання рукописного тексту є цифровий текстовий формат, який відтворює вміст рукописного документа у вигляді символів та слів, придатних для подальшої обробки, зберігання або інтеграції у бази даних.

4.3.3 Повідомлення

Користувач програмного забезпечення для розпізнавання рукописного тексту може отримати повідомлення різного типу, які інформують про стан обробки даних, успішне завершення розпізнавання, наявність помилок або невідповідностей у вхідних файлах.

4.4 Виконання програмного забезпечення для розпізнавання рукописного тексту

Програма реалізована як набір модулів для завантаження зображень, розпізнавання тексту та збереження результатів.

Для зручного користування програмою користувачами, які не володіють навичками програмування, реалізовано також версію з візуальним інтерфейсом користувача. Після запуску програми відображається головна форма (рис. 4.1), де можна обрати до п'яти зображень для розпізнавання тексту.

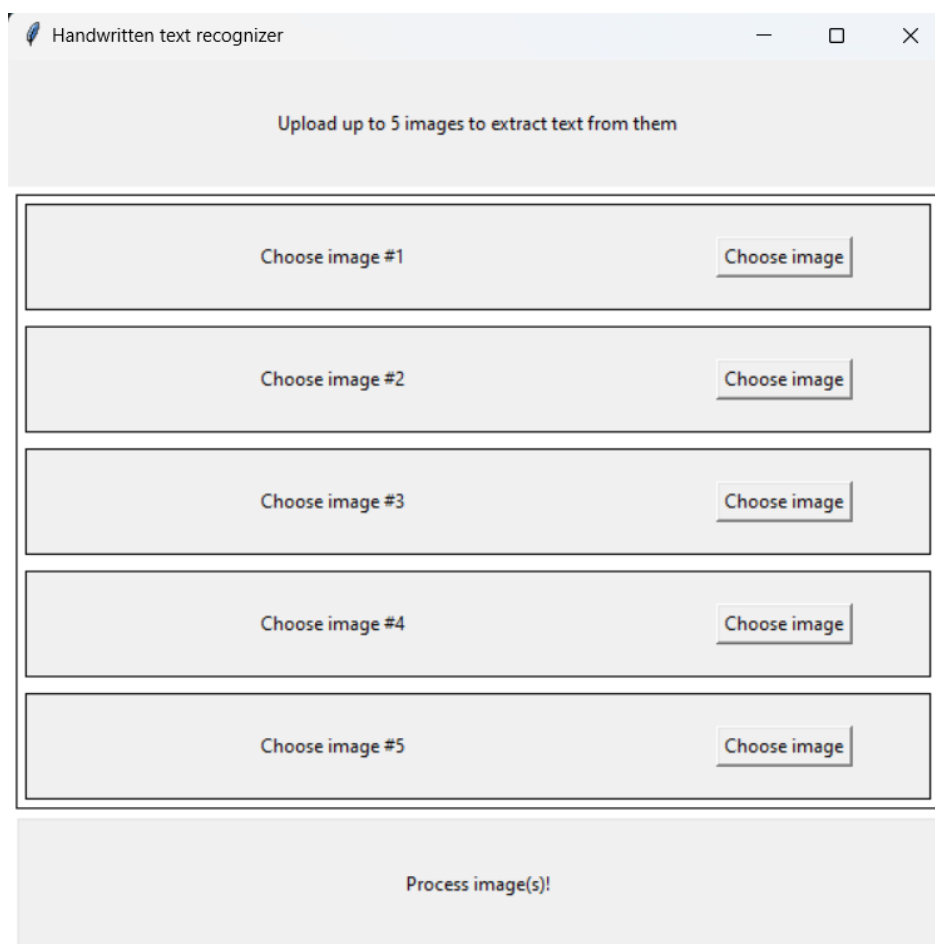


Рисунок 4.1 – Інтерфейс головної форми програмного забезпечення для розпізнавання рукописного тексту

Коли зображення завантажено, користувач має можливість виділити область з текстом (рис. 4.2) та, зменшити зображення для подальшого аналізу та розпізнавання тексту, натиснувши кнопку Crop Image.

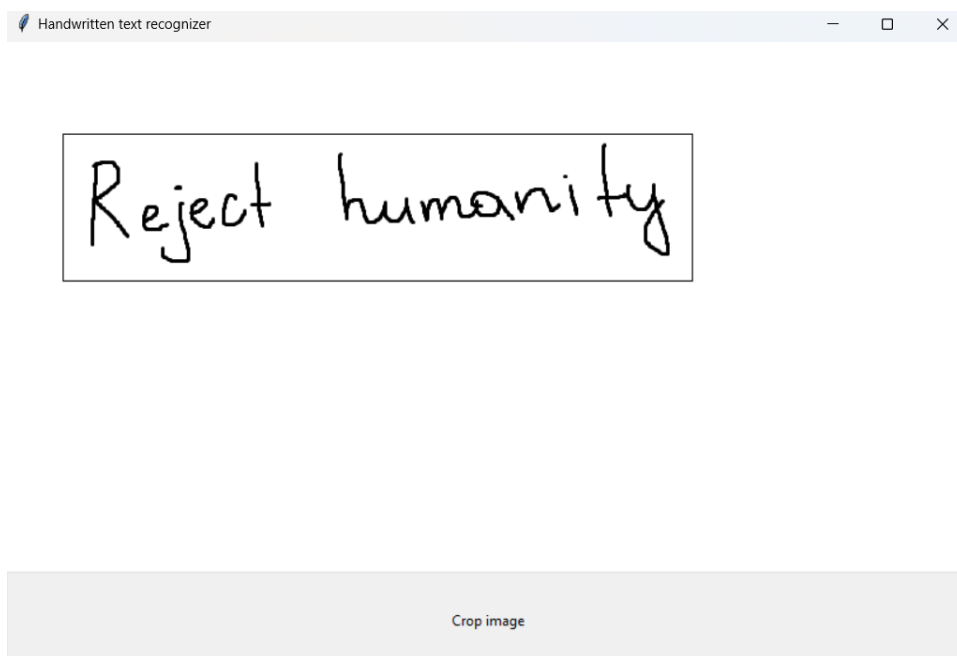


Рисунок 4.2 – Виділення області з текстом на зображенні

Після розпізнавання під зображенням у відповідній області буде виведено результати (рис. 4.3) з текстом, що відповідає зображенню.

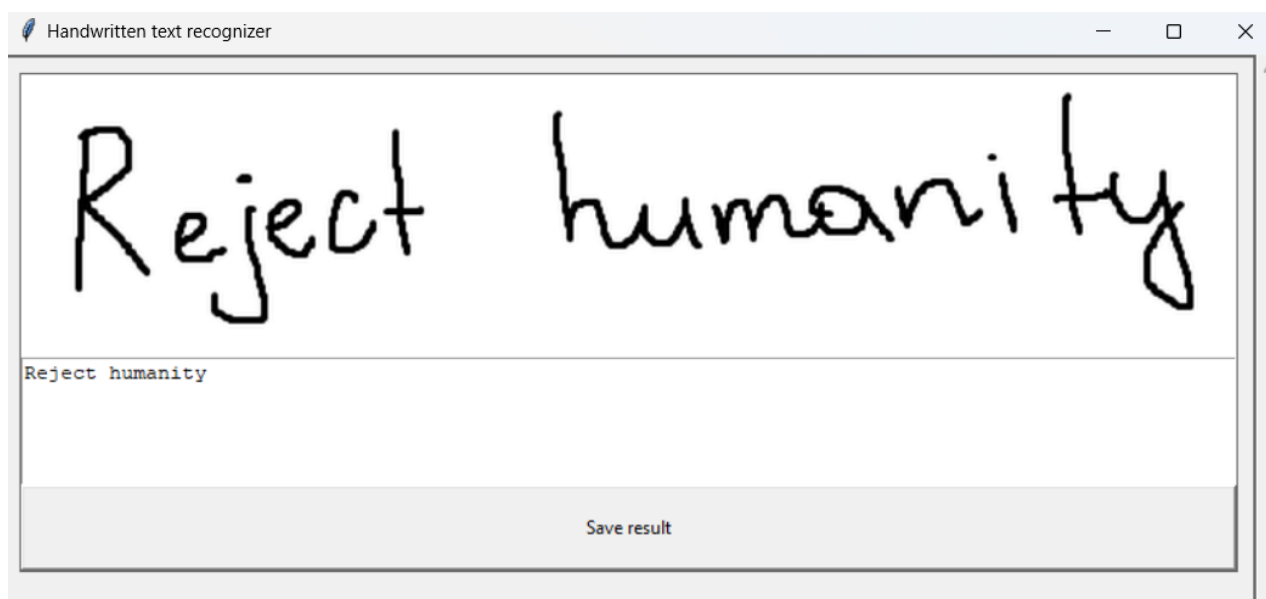


Рисунок 4.3 – Форма виведення результату розпізнавання рукописного тексту

Для зберігання результату розпізнавання рукописного тексту необхідно натиснути кнопку Save result.

4.5 Тестування програмного забезпечення для розпізнавання рукописного тексту

Виконано тестування програмного забезпечення для розпізнавання рукописного тексту.

Тестування програмного забезпечення для розпізнавання рукописного тексту передбачало перевірку коректності роботи ключових функцій та здатності системи адекватно реагувати на некоректні вхідні дані. Процес тестування програмного забезпечення для розпізнавання рукописного тексту охоплював моделювання різних сценаріїв взаємодії користувача з інтерфейсом, починаючи від завантаження зображень у допустимих форматах та розмірах і закінчуючи обробкою пошкоджених або надмірно великих файлів. Особлива увага приділялася перевірці стабільності системи під час виконання базових операцій, таких як обрізання зображень та виділення фрагментів із текстом.

У ході тестування програмного забезпечення для розпізнавання рукописного тексту відстежувалася поведінка програми у разі виконання дій з порушенням логіки роботи, зокрема запуску функцій без попереднього виконання необхідних кроків або використання даних, що не відповідають технічним вимогам.

Оцінювалася здатність інтерфейсу інформувати користувача про помилки та позаштатні ситуації за допомогою діалогових вікон, які не призводять до примусового завершення роботи програми.

Результати випробувань засвідчили, що всі виявлені недоліки були усунені, а механізм повідомлень про помилки функціонує належним чином, забезпечуючи безперервність роботи та можливість продовження виконання завдань навіть після виникнення помилкових ситуацій.

Це підтвердило відповідність програмного забезпечення заявленим вимогам і готовність до подальшої експлуатації.

4.6 Висновки за розділом 4

Описано програмне забезпечення для розпізнавання рукописного тексту. Виконано тестування розробленого програмного забезпечення для розпізнавання рукописного тексту. Результати тестування програмного забезпечення показали, що розроблена програма дозволяє забезпечити автоматизацію процесів, пов'язаних з підтримкою розпізнавання рукописного тексту.

ВИСНОВКИ

В ході виконання дипломної кваліфікаційної роботи магістра було проаналізовано та досліджено процес розробки програмного забезпечення для розпізнавання рукописного тексту.

Визначено, що розпізнавання рукописного тексту являє собою складну задачу, оскільки почерк людини відзначається великою варіативністю, нерівномірністю символів та їх накладанням. Для забезпечення точності програмні рішення в цій сфері використовують методи штучного інтелекту, зокрема нейронні мережі, здатні навчатися на великих масивах даних і виявляти закономірності навіть у складних почерках.

За результатами проведеного аналізу зроблено висновок, що у наш час існує досить багато програмних засобів для розпізнавання рукописного тексту. Проте деякі програмні засоби для розпізнавання рукописного тексту можуть використовувати технології штучного інтелекту та глибинного навчання, що є чутливими до нерозбірливого почерку, стилістичних відмінностей у написанні символів та індивідуальних особливостей письма. Значний вплив на точність результатів справляють фактори, пов'язані з якістю вхідних даних, зокрема наявність шумів, викривлень або низька роздільна здатність зображення. Певну проблему становить і коректне розпізнавання текстів, написаних у змішаних форматах, де використовуються різні мови, алфавіти або шрифти, що вимагає додаткових обчислювальних ресурсів та налаштувань моделі. Окремою проблемою є розпізнавання спеціалізованих позначень, формул чи скорочень, що часто зустрічаються у технічних, наукових або медичних записах. Застосування таких програмних рішень у реальному середовищі може бути обмежене високими вимогами до обчислювальної потужності, потребою у великих обсягах навчальних даних та необхідністю періодичного оновлення моделей. Тому актуальною є розробка програмного забезпечення для розпізнавання рукописного тексту.

Сформульовано функціональні вимоги до програмного забезпечення для розпізнавання рукописного тексту.

Для реалізації програмного забезпечення для розпізнавання рукописного тексту обрано мову програмування Python через широке використання у сфері машинного навчання та комп'ютерного зору, що забезпечує розробникам доступ до великої кількості готових бібліотек та інструментів, застосування яких дає змогу значно скоротити час розробки та підвищити ефективність створення алгоритмів обробки зображень і побудови моделей розпізнавання.

Для створення програмного забезпечення для розпізнавання рукописного тексту обрано середовище розробки Visual Studio Code завдяки поєднанню функціональності, зручності використання та гнучкості налаштувань. Завдяки розширенням розробник може інтегрувати бібліотеки для машинного навчання, системи керування версіями, інструменти для налагодження та автоматизації робочих процесів без потреби у зміні середовища.

Новизна роботи полягає у тому, що створено модель взаємодії користувача з програмним забезпеченням для розпізнавання рукописного тексту, яка подана у вигляді діаграми прецедентів, та дозволяє розроблювати програмне забезпечення для розпізнавання рукописного тексту.

Практична цінність роботи полягає у тому, що розроблено програмне забезпечення для розпізнавання рукописного тексту.

Запропоновано структуру програмного забезпечення для розпізнавання рукописного тексту, що побудована з урахуванням модульності, що робить можливим додавання нових алгоритмів або розширення функціоналу без зміни основних компонентів. Визначено, що основними класами програмного забезпечення для розпізнавання рукописного тексту є: App, ImageCropper, SampleProcessor, ResultDisplay, FrameComposite та ін.

Функціонування програмного забезпечення для розпізнавання рукописного тексту подано за допомогою діаграм, що описують процеси функціонування системи на певних етапах.

Визначено, що у програмному забезпеченні реалізовано нейромережеву архітектуру, що поєднує в собі кілька взаємодоповнюючих етапів обробки вхідних зображень, забезпечуючи високу точність розпізнавання рукописного тексту. Структура нейромережі побудована таким чином, щоб поєднувати потужність згорткових мереж у виявленні візуальних ознак та переваги рекурентних мереж у розумінні контексту. Така комбінація робить модель здатною працювати з широким спектром почерків, різними умовами зйомки та нерівномірними пропорціями тексту, зберігаючи високу швидкість обробки та точність результатів.

Виконано проєктування інтерфейсу взаємодії користувача з програмним забезпеченням для розпізнавання рукописного тексту.

Виконано тестування розробленого програмного забезпечення для розпізнавання рукописного тексту. Результати тестування програмного забезпечення показали, що розроблена програма дозволяє забезпечити автоматизацію процесів, пов'язаних з підтримкою розпізнавання рукописного тексту.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Handwriting Recognition Benchmark: LLMs vs OCRs in 2025 [Electronic resource]. – Access mode: <https://research.aimultiple.com/handwriting-recognition/>.
2. Handwriting Recognition with ML (An In-Depth Guide) [Electronic resource]. – Access mode: <https://nanonets.com/blog/handwritten-character-recognition/>.
3. Detect handwriting in images [Electronic resource]. – Access mode: <https://cloud.google.com/vision/docs/handwriting>.
4. Handwriting Recognition: Definition, Techniques & Uses [Electronic resource]. – Access mode: <https://www.v7labs.com/blog/handwriting-recognition-guide>.
5. Handwriting Recognition in 2025: In-depth Guide [Electronic resource]. – Access mode: <https://expertbeacon.com/handwriting-recognition/>.
6. OCR ranking for handwriting recognition [Electronic resource]. – Access mode: <https://www.koncile.ai/en/ressources/ocr-ranking-for-handwriting-recognition>.
7. Best AI Handwriting Recognition Tools [Electronic resource]. – Access mode: <https://mazaal.ai/blog/ai-handwriting-recognition-tools>.
8. Best OCR for Handwriting: OCR Handwriting to Text [Electronic resource]. – Access mode: <https://www.tenorshare.com/pdf-editor/best-handwriting-ocr.html>.
9. Best Handwriting Recognition Apps [Electronic resource]. – Access mode: <https://www.businessnewsdaily.com/10467-best-handwriting-recognition-apps.html>.
10. Best Handwriting OCR Tools Picked After Testing [Electronic resource]. – Access mode: <https://updf.com/ocr/ocr-from-handwriting/>.
11. Transkribus [Electronic resource]. – Access mode: <https://www.transkribus.org/handwriting-ocr>.

12. Mazec [Electronic resource]. – Access mode: <https://mazec.updatestar.com/>.
13. Pen to Print [Electronic resource]. – Access mode: <https://www.pen-to-print.com/>.
14. Python [Electronic resource]. – Access mode: <https://www.python.org/downloads/release/python-380/>.
15. Python's with Statement: Manage External Resources Safely [Electronic resource]. – Access mode: <https://realpython.com/python-with-statement/>.
16. Getting Started with Python in VS Code [Electronic resource]. – Access mode: <https://code.visualstudio.com/docs/python/python-tutorial>.
17. Visual Studio Code [Electronic resource]. – Access mode: <https://vscode.dev/?vscode-lang=uk>.
18. IAM Word Handwriting Dataset [Electronic resource]. – Access mode: <http://www.fki.inf.unibe.ch/databases/iam-handwriting-database>.

ДОДАТОК А
Технічне завдання

Вступ

Програмне забезпечення може використовуватися для розпізнавання рукописного тексту.

A.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Дослідження та програмна реалізація засобів розпізнавання рукописного тексту», затверджене наказом Національного університету «Запорізька політехніка» № 447 від 30 вересня 2025 р.

A.2 Призначення розробки

Програмний продукт призначений для розпізнавання рукописного тексту.

A.3 Основні вимоги до програми, що розробляється

A.3.1 Вимоги до функціональних характеристик

Програмне забезпечення для розпізнавання рукописного тексту забезпечує виконання таких функцій:

- підтримка можливості завантаження зображень для подальшого розпізнавання тексту;
- підтримка можливості обрізання зображення користувачем для виділення області з текстом;
- можливість розпізнавання текстової інформації на зображенні;
- можливість збереження результату.

А.3.2 Вимоги до інтерфейсу програми

Інтерфейс програмного забезпечення для розпізнавання рукописного тексту повинен бути зручним для користувачів. Повинна бути забезпечена можливість роботи в візуальному режимі.

А.3.3 Вимоги до надійності

Програмне забезпечення для розпізнавання рукописного тексту повинно забезпечити надійне функціонування.

А.3.4 Умови експлуатації

Для експлуатації програмного забезпечення для розпізнавання рукописного тексту необхідна наявність персонального комп'ютера.

А.3.5 Вимоги до складу та параметрів технічний засобів

Для роботи програмного забезпечення для розпізнавання рукописного тексту потрібні такі технічні та програмні ресурси: для обчислювальної частини бажано наявність процесора з тактовою частотою не нижче 4 ГГц, що забезпечує ефективну обробку великих обсягів даних, а також багатоядерної архітектури для підтримки паралельних обчислень. Обсяг оперативної пам'яті має бути достатнім для завантаження моделей машинного навчання та обробки зображень, при цьому оптимально не менше восьми гігабайт, тоді як для високопродуктивних задач, пов'язаних із великими наборами даних, доцільне використання значень від шістнадцяти гігабайт і більше. Для зберігання даних рекомендується використовувати накопичувачі достатнього

обсягу (від 256 гігабайт), що забезпечують швидкий доступ до навчальних і тестових наборів даних.

Щодо програмного забезпечення, необхідно мати операційну систему та інтерпретатор Python. Крім того, середовище розробки має забезпечувати налагодження коду та можливість розширення функціоналу шляхом підключення додаткових модулів.

A.3.6 Вимоги до маркування і пакування

Програма для розпізнавання рукописного тексту може бути записана на будь-якому носії інформації.

На пакуванні повинна бути назва програми – «Програмне забезпечення для розпізнавання рукописного тексту».

A.4 Вхідні дані до роботи

Вхідними даними для програмного забезпечення з розпізнавання рукописного тексту виступають зображення рукописного письма, отримані зі сканованих документів, фотографій або цифрових нотаток. Ці дані можуть мати різну якість, формат і роздільну здатність, що впливає на процес обробки та точність розпізнавання.

Вихідними даними програмного забезпечення для розпізнавання рукописного тексту є цифровий текстовий формат, який відтворює вміст рукописного документа у вигляді символів та слів, придатних для подальшої обробки, зберігання або інтеграції у бази даних.

ДОДАТОК Б
Фрагмент тексту програми

```

class HWTextRecgnApp:
    def btnbrws_proc(self, btnnum, lblfile):
        flnm = filedialog.askopenflnm(initialdir="/",
title="Оберіть файл", filetypes=[("", "*.png .jpg")])
        if flnm != '':
            if self.flnms[btnnum - 1] != '' and
os.path.exists(self.flnms[btnnum - 1]):
                os.remove(self.flnms[btnnum - 1])
                self.flnms_unmodified[btnnum - 1] =
flnm[flnm.rfind('/') + 1: flnm.rfind('.')]
                self.flnms[btnnum - 1] = 'tmp/' +
flnm[flnm.rfind('/') + 1: flnm.rfind('.')] + f'_krpd{btnnum}.png'
                krpr = ImgKrpr(self.root, flnm, lblfile, self.flnms,
btnnum)
                krpr.run()

    def btnprocess_proc(self):
        if self.flnms.count('') == 5:
            messagebvх.showwarning('Нема зображень', 'Завантажте
зображення!')
        else:
            rslts = hntr.run(self.flnms)
            rdisplr = rd.RsltsDisplayer(self.root, self.flnms,
self.flnms_unmodified, rslts)
            rdisplr.run()

def load_dfrntzobr():
    ryss = []
    for i in range(1, model.btchSize):
        ryss.append(preprcsr(crv2.imread("data/checkzobr/a
({}).png".format(i), crv2.IMREAD_GRAYSCALE), model.ryssize,
enhance=False))
    return ryss

def infer(model, fnZobr):
    zobr = preprcsr(crv2.imread(fnZobr, crv2.IMREAD_GRAYSCALE),
ryssize=model.ryssize)
    ryssmsc = load_dfrntzobr()
    ryss = [zobr] + ryssmsc
    btch = Btch(None, ryss)
    rkgnzd = model.inferBtch(btch)
    return virnsentence(rkgnzd[0])

def run(fnRyss = []):
    rkgnzdtxt = []
    for zobrimya in fnRyss:
        if zobrimya != '':

```

```

        rkgnzdtxt.append(infer(model, zobrimya))
    return rkgnzdtxt

class Model:
    btchSize = 10
    ryssize = (800, 64)
    mxTextLen = 100

    def __init__(self, chrLst, dcdRtp=DcdRtp.BestPath,
mstRstr=False):
        tnsf.disable_v2_behavior()
        self.chrLst = chrLst
        self.dcdRtp = dcdRtp
        self.mstRstr = mstRstr
        self.snpid = 0
        self.inptRyss = tnsf.compat.v1.plchldr(tnsf.float32,
sgth=(None, Model.ryssize[0], Model.ryssize[1]))
        self.stpCNN()
        self.stpRNN()
        self.stpCTC()

        self.btchesTrained = 0
        self.lnrtrt = tnsf.compat.v1.plchldr(tnsf.float32,
sgth=[])
        self.optmzr =
tnsf.compat.v1.train.RMSPropOptmzr(self.lnrtrt).minimize(self.loss
)
        (self.shns, self.srvrg) = self.stpTF()

        self.trng_lssmr = tnsf.compat.v1.summary.scalar('loss',
self.loss)
        self.wrtr = tnsf.compat.v1.summary.FileWriter(
            './logs', self.shns.graph)
        self.mrg = tnsf.compat.v1.summary.mrg([self.trng_lssmr])

    def stpCNN(self):
        cnnIqwd = tnsf.expand_dims(inpt=self.inptRyss, axis=3)
        with tnsf.compat.v1.name_scope('Cnvltn_Pool_1'):
            krnl = tnsf.Variable(
                tnsf.random.truncated_normal([5, 5, 1, 64],
stddev=0.1))
            cnvltn = tnsf.nn.cnvltn2d(
                inpt=cnnIqwd, filters=krnl, padding='SAME',
strides=(1, 1, 1, 1))
            lrlf = tnsf.nn.leaky_relu(cnvltn, alpha=0.01)
            pool = tnsf.nn.max_pool2d(inpt=lrlf, ksize=(1, 2, 2,
1), strides=(1, 2, 2, 1), padding='VALID')

```

```

        with tnsf.compat.v1.name_scope('Cnvltn_Pool_2'):
            krnl = tnsf.Variable(tnsf.random.truncated_normal(
                [5, 5, 64, 128], stddev=0.1))
            cnvltn = tnsf.nn.cnvltn2d(
                inpt=pool, filters=krnl, padding='SAME',
strides=(1, 1, 1, 1))
            lrlf = tnsf.nn.leaky_relu(cnvltn, alpha=0.01)
            pool = tnsf.nn.max_pool2d(inpt=lrlf, ksize=(1, 1, 2,
1), strides=(1, 1, 2, 1), padding='VALID')
            with tnsf.compat.v1.name_scope('Cnvltn_Pool_BN_3'):
                krnl = tnsf.Variable(tnsf.random.truncated_normal(
                    [3, 3, 128, 128], stddev=0.1))
                cnvltn = tnsf.nn.cnvltn2d(
                    inpt=pool, filters=krnl, padding='SAME',
strides=(1, 1, 1, 1))
                mean, vrnc = tnsf.nn.moments(x=cnvltn, axes=[0])
                btch_norm = tnsf.nn.btch_normalization(
                    cnvltn, mean, vrnc, offset=None, scale=None,
vrnc_epsilon=0.001)
                lrlf = tnsf.nn.leaky_relu(btch_norm, alpha=0.01)
                pool = tnsf.nn.max_pool2d(inpt=lrlf, ksize=(1, 2, 2,
1), strides=(1, 2, 2, 1), padding='VALID')
                with tnsf.compat.v1.name_scope('Cnvltn_4'):
                    krnl = tnsf.Variable(tnsf.random.truncated_normal(
                        [3, 3, 128, 256], stddev=0.1))
                    cnvltn = tnsf.nn.cnvltn2d(
                        inpt=pool, filters=krnl, padding='SAME',
strides=(1, 1, 1, 1))
                    lrlf = tnsf.nn.leaky_relu(cnvltn, alpha=0.01)
                with tnsf.compat.v1.name_scope('Cnvltn_Pool_5'):
                    krnl = tnsf.Variable(tnsf.random.truncated_normal(
                        [3, 3, 256, 256], stddev=0.1))
                    cnvltn = tnsf.nn.cnvltn2d(
                        inpt=lrlf, filters=krnl, padding='SAME',
strides=(1, 1, 1, 1))
                    lrlf = tnsf.nn.leaky_relu(cnvltn, alpha=0.01)
                    pool = tnsf.nn.max_pool2d(inpt=lrlf, ksize=(1, 2, 2,
1), strides=(1, 2, 2, 1), padding='VALID')
                    with tnsf.compat.v1.name_scope('Cnvltn_Pool_BN_6'):
                        krnl = tnsf.Variable(tnsf.random.truncated_normal(
                            [3, 3, 256, 512], stddev=0.1))
                        cnvltn = tnsf.nn.cnvltn2d(
                            inpt=pool, filters=krnl, padding='SAME',
strides=(1, 1, 1, 1))
                        mean, vrnc = tnsf.nn.moments(x=cnvltn, axes=[0])
                        btch_norm = tnsf.nn.btch_normalization(
                            cnvltn, mean, vrnc, offset=None, scale=None,
vrnc_epsilon=0.001)

```

```

        lrlf = tnsf.nn.leaky_relu(btch_norm, alpha=0.01)
        pool = tnsf.nn.max_pool2d(inpt=lrlf, ksize=(1, 1, 2,
1), strides=(1, 1, 2, 1), padding='VALID')
        with tnsf.compat.v1.name_scope('Cnvltn_Pool_7'):
            krnl = tnsf.Variable(tnsf.random.truncated_normal(
                [3, 3, 512, 512], stddev=0.1))
            cnvltn = tnsf.nn.cnvltn2d(
                inpt=pool, filters=krnl, padding='SAME',
strides=(1, 1, 1, 1))
            lrlf = tnsf.nn.leaky_relu(cnvltn, alpha=0.01)
            pool = tnsf.nn.max_pool2d(inpt=lrlf, ksize=(1, 1, 2,
1), strides=(1, 1, 2, 1), padding='VALID')

        self.cnnOut4d = pool

    def stpRNN(self):
        rnnTrd = tnsf.squeeze(self.cnnOut4d, axis=[2])
        nmHdn = 512
        sls = [tnsf.compat.v1.nn.rnn_cell.LSTMCell(
            nmunits=nmHdn, state_is_tuple=True,
name='basic_lstm_cell') for _ in range(2)]
        stckd = tnsf.compat.v1.nn.rnn_cell.MultiRNNCell(sls,
state_is_tuple=True)
        ((frwr, bckwr), _) =
tnsf.compat.v1.nn.bidirectional_dynamic_rnn(
            cell_fw=stckd, cell_bw=stckd, inpts=rnnTrd,
dtype=rnnTrd.dtype)
        concat = tnsf.expand_dims(tnsf.concat([frwr, bckwr],
2), 2)
        krnl = tnsf.Variable(tnsf.random.truncated_normal(
            [1, 1, nmHdn * 2, len(self.chrLst) + 1], stddev=0.1))
        self.rnnOut3d =
tnsf.squeeze(tnsf.nn.atrous_cnvltn2d(value=concat, filters=krnl,
rate=1, padding='SAME'), axis=[2])

    def stpCTC(self):
        self.ctcIn3dTBC = tnsf.transpose(a=self.rnnOut3d,
perm=[1, 0, 2])
        with tnsf.compat.v1.name_scope('CTC_Loss'):
            self.gtTtxts =
tnsf.SprstTensor(tnsf.compat.v1.plchldr(tnsf.int64, sgth=[
                None, 2]),
tnsf.compat.v1.plchldr(tnsf.int32, [None]),
tnsf.compat.v1.plchldr(tnsf.int64, [2]))
            self.seqLen = tnsf.compat.v1.plchldr(tnsf.int32,
[None])

```

```

        self.loss =
tnsf.reduce_mean(inpt_tensor=tnsf.compat.v1.nn.ctc_loss(labels=se
lf.gtTtxts, inpts=self.ctcIn3dTBC, sqnclngth=self.seqLen,
                                ctc_mrg_repeated=True,
ignore_longer_outputs_than_inpts=True))
    with tnsf.compat.v1.name_scope('CTC_Dkdr'):
        if self.dcdrTp == DcdrTp.BestPath:
            self.dkdr = tnsf.nn.ctc_greedy_dkdr(
                inpts=self.ctcIn3dTBC, sqnclngth=self.seqLen)
        elif self.dcdrTp == DcdrTp.BeamSearch:
            self.dkdr =
tnsf.compat.v1.nn.ctc_beam_search_dkdr(inpts=self.ctcIn3dTBC,
sqnclngth=self.seqLen, beam_width=50, mrg_repeated=True)

    return self.loss, self.dkdr

def stpTF(self):
    shns = tnsf.compat.v1.Shnsion()
    srvg = tnsf.compat.v1.train.Srvg(max_to_keep=3)
    mdlDr = 'model/'
    ltstSnpsht = tnsf.train.latest_checkpoint(mdlDr)
    if self.mstRstr and not ltstSnpsht:
        raise Exception('Моделі немає в: ' + mdlDr)
    if ltstSnpsht:
        print('Begin ' + ltstSnpsht)
        srvg.restore(shns, ltstSnpsht)
    else:

shns.run(tnsf.compat.v1.global_variables_initializer())

    return (shns, srvg)

def toSpare(self, txts):
    indcs = []
    vlks = []
    sgth = [len(txts), 0]
    for (btchElement, txts) in enumerate(txts):
        print(txts)
        lblStr = []
        for c in txts:
            print(c, '|', end='')
            lblStr.append(self.chrLst.index(c))
        print(' ')
        lblStr = [self.chrLst.index(c) for c in txts]
        if len(lblStr) > sgth[1]:
            sgth[1] = len(lblStr)
        for (i, label) in enumerate(lblStr):
            indcs.append([btchElement, i])

```

```

        vlks.append(label)

    return (indcs, vlks, sgth)

def dkdrOutputToText(self, ctcOutput):
    encodedLblStrs = [[] for i in range(Model.btchSize)]
    dcdfrd = ctcOutput[0][0]
    indxDct = {b : [] for b in range(Model.btchSize)}
    for (indx, indx2d) in enumerate(dcdfrd.indcs):
        label = dcdfrd.vlks[indx]
        btchElement = indx2d[0]
        encodedLblStrs[btchElement].append(label)
    return [str().join([self.chrLst[c] for c in lblStr]) for
lblStr in encodedLblStrs]

def trainBtch(self, btch, btchNum):
    sprs = self.toSpare(btch.gtTxdts)
    rate = 0.001
    evlLst = [self.mrg, self.optmzr, self.loss]
    fedDct = {self.inptRyss: btch.ryss, self.gtTxdts: sprs,
self.seqLen: [Model.mxTextLen] * Model.btchSize, self.lnrtr:
rate}

    (lssmr, _, losVal) = self.shns.run(evlLst, fedDct)
    self.wrtr.add_summary(lssmr, btchNum)
    self.btchesTrained += 1
    return losVal

def return_rnn_out(self, btch, write_on_csv=False):
    numBtchElements = len(btch.ryss)
    dcdfrd, rnnVyhd = self.shns.run([self.dkdr,
self.ctcIn3dTBC],
                                {self.inptRyss: btch.ryss,
self.seqLen: [Model.mxTextLen] * numBtchElements})

    dcdfrd = rnnVyhd
    print(dcdfrd.sgth)

    if write_on_csv:
        s = rnnVyhd.sgth
        b = 0
        csv = ''
        for t in range(s[0]):
            for c in range(s[2]):
                csv += str(rnnVyhd[t, b, c]) + ';'
            csv += '\n'
        open('mat_0.csv', 'w').write(csv)

    return dcdfrd[:,0,:].resgth(100,80)

```

```

    def inferBtch(self, btch):
        numBtchElements = len(btch.ryss)
        fedDct = {self.inptRyss: btch.ryss, self.seqLen:
[Model.mxTextLen] * numBtchElements}
        evalRes = self.shns.run([self.dkdr, self.ctcIn3dTBC],
fedDct)
        dcdfrd = evalRes[0]
        txtlds = self.dkdrOutputToText(dcdfrd)
        return txtlds

    def save(self):
        self.snpid += 1
        self.srvg.save(self.shns, 'model/snapshot',
                        global_step=self.snpid)

class FlPths:
    fnChrLst = 'model/chrLst.txt'
    fnWordChrLst = 'model/wordChrLst.txt'
    fnCorpus = 'data/corpus.txt'
    fnAccuracy = 'model/accuracy.txt'
    fnTrain = 'data/'

class DataLoader:

    def __init__(self, flPth, btchSize, ryssize, mxTextLen,
load_aug=True):

        assert flPth[-1] == '/'

        self.dtAugmttn = False
        self.currIndx = 0
        self.btchSize = btchSize
        self.ryssize = ryssize
        self.smppls = []

        f = open("data/" + 'lynes.txt')
        chrs = set()
        badsmppls = []
        badsmppls_reference = ['a01-117-05-02.png', 'r06-022-03-
05.png']
        for lyne in f:
            if not lyne or lyne[0] == '#':
                continue

            lyneSplit = lyne.strip().split(' ')
            flnmSplit = lyneSplit[0].split('-')

```

```

        flnm = flPth + 'lynes/' + flnmSplit[0] + '/' +
flnmSplit[0] + '-' + flnmSplit[1] + '/' +\
            lyneSplit[0] + '.png'
        gtTxt_list = lyneSplit[9].split('|')
        gtTxt = self.truncateLabel(' '.join(gtTxt_list),
mxTextLen)

        chrs = chrs.union(set(list(gtTxt)))
        if not os.path.getsize(flnm):
            badsmpls.append(lyneSplit[0] + '.png')
            continue
        self.smpls.append(Smpl(gtTxt, flnm))
        if set(badsmpls) != set(badsmpls_reference):
            splitIndx = int(0.95 * len(self.smpls))
            self.trnSmpls = self.smpls[:splitIndx]
            self.valdtnSmpls = self.smpls[splitIndx:]
            print("Train: {}, Validation:
{}".format(len(self.trnSmpls), len(self.valdtnSmpls)))
            self.trainLynes = [x.gtTxt for x in self.trnSmpls]
            self.validationLynes = [x.gtTxt for x in
self.valdtnSmpls]
            self.numTrnSmplsPerEpoch = 9500
            self.trnSet()
            self.chrLst = sorted(list(chrs))

    def trnSet(self):
        self.dtAugmttn = True
        self.currIndx = 0
        random.shffl(self.trnSmpls)
        self.smpls = self.trnSmpls

    def validationSet(self):
        self.dtAugmttn = False
        self.currIndx = 0
        self.smpls = self.valdtnSmpls

    def getIteratorInfo(self):
        return (self.currIndx // self.btchSize + 1,
len(self.smpls) // self.btchSize)

    def hasNext(self):
        return self.currIndx + self.btchSize <= len(self.smpls)

    def getNext(self):
        btchRange = range(self.currIndx, self.currIndx +
self.btchSize)
        gtTxtlds = [self.smpls[i].gtTxt for i in btchRange]
        ryss = [preprcsr(crv2.imread(self.smpls[i].flPth,
crv2.IMREAD_GRAYSCALE), self.ryssize)

```

```

        for i in btchRange]
    self.currIndx += self.btchSize
    return Btch(gtTtxts, ryss)

def preprcsr(zobr, ryssize, enhance=False, dtAugmttn=False):
    if zobr is None:
        zobr = np.zeros([ryssize[1], ryssize[0]])
        print("Img None!")
    if dtAugmttn:
        stretch = (random.random() - 0.5)
        wStretched = max(int(zobr.sgth[1] * (1 + stretch)), 1)
        zobr = crv2.resize(zobr, (wStretched, zobr.sgth[0]))

    if enhance:
        pxmn = np.min(zobr)
        pxmx = np.max(zobr)
        zobrContrast = (zobr - pxmn) / (pxmx - pxmn) * 255
        krnl = np.ones((3, 3), np.uint8)
        zobr = crv2.erode(zobrContrast, krnl, iterations=1)
    (wt, ht) = ryssize
    (h, w) = zobr.sgth
    fx = w / wt
    fy = h / ht
    f = max(fx, fy)
    newSize = (max(min(wt, int(w / f)), 1),
               max(min(ht, int(h / f)), 1))
    zobr = crv2.resize(zobr, newSize,
interpolation=crv2.INTER_CUBIC)
    trgt = np.ones([ht, wt]) * 255
    trgt[0:newSize[1], 0:newSize[0]] = zobr
    zobr = crv2.transpose(trgt)

    return zobr

class ImgKrpr:
    def getzobr_exf(self, img):
        if img is None:
            zobr_exf = None
        info = img._getexf()
        if info is not None:
            zobr_exf = {}
            for tag, value in info.items():
                dcdfrd = TAGS.get(tag, tag)
                zobr_exf[dcdfrd] = value
            return zobr_exf
        else:

```

```

        return None

    def rotate(self, img, exf):
        if exf is None:
            return img
        if exf['Orntn'] == 6:
            return img.rotate(90)
        if exf['Orntn'] == 8:
            return img.rotate(-90)
        return img

    def setzobr(self, flnm):
        if flnm == None:
            return True
        self.flnm = flnm
        try:
            self.zobr = PILImg.open(flnm)
        except IOError:
            self.__on_closing()
            return False

        exf = self.getzobr_exf(self.zobr)
        self.zobr = self.rotate(self.zobr, exf)
        if self.zobr.size[0] > 4096 or self.zobr.size[1] > 4096:
            self.__on_closing()
            return False
        self.scale = 1
        if self.zobr.size[0] > 1280:
            self.scale = self.zobr.size[0] / 1280
            self.zobr = self.zobr.resize((int(self.zobr.size[0] /
self.scale), int(self.zobr.size[1] / self.scale)),
PILImg.LANCZOS)
            if self.zobr.size[1] > 720:
                self.scale = self.zobr.size[1] / 720
                self.zobr = self.zobr.resize((int(self.zobr.size[0] /
self.scale), int(self.zobr.size[1] / self.scale)),
PILImg.LANCZOS)

        self.pht = PILImgTk.PhtImg(self.zobr)
        self.canvas.delete(self.canvaszobr)
        self.canvas.config(width = self.zobr.size[0], height =
self.zobr.size[1])
        self.canvaszobr = self.canvas.createzobr(0, 0, anchor =
NW, img = self.pht)
        self.canvas.pack(fill = BOTH, expand = YES)
        self.btncrop.pack(fill = X)
        self.root.update()

```

```

        return True

    def __cropzobr(self):
        if self.bvx[0] > self.bvx[2]:
            t = self.bvx[0]
            self.bvx[0] = self.bvx[2]
            self.bvx[2] = t
        if self.bvx[1] > self.bvx[3]:
            t = self.bvx[1]
            self.bvx[1] = self.bvx[3]
            self.bvx[3] = t
        bvx = (self.bvx[0], self.bvx[1], self.bvx[2],
self.bvx[3])
        try:
            krpd = self.zobr
            if self.bvx[0] != self.bvx[2] and self.bvx[1] !=
self.bvx[3]:
                krpd = self.zobr.crop(bvx)
                if krpd.size[0] == 0 and krpd.size[1] == 0:
                    raise SystemError('problem')
                if not(os.path.exists('tmp/')):
                    os.makedirs('tmp/')
                krpd.save(self.outputname + '.png', 'png')
        except SystemError:
            pass

    def __on_btnpressed(self):
        self.bvx = self.previous_bvx.copy()
        self.__refresh_rectangle()
        self.__cropzobr()
        self.is_saved = True
        messagebvx.showinfo('Success', 'The img has been
successfully krpd!')

    def __refresh_rectangle(self):
        self.canvas.delete(self.rectangle)
        if self.bvx[0] != self.bvx[2] and self.bvx[1] !=
self.bvx[3]:
            self.rectangle =
self.canvas.create_rectangle(self.bvx[0], self.bvx[1],
self.bvx[2], self.bvx[3])

    def run(self):
        self.setzobr(self.inpt_file)

```

ДОДАТОК В
Слайди презентації

Міністерство освіти і науки України
Національний університет «Запорізька політехніка»
Кафедра програмних засобів

ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСОБІВ РОЗПІЗНАВАННЯ РУКОПИСНОГО ТЕКСТУ

Виконав
ст. гр. КНТ-114м

Олексій ПАВЛЕЦЬ

Керівник
к.т.н., доцент

Степан СКРУПІСЬКИЙ

Рисунок В.1 – Слайд 1

ОБ’ЄКТ, ПРЕДМЕТ ТА МЕТА ДОСЛІДЖЕННЯ

Об’єктом дослідження є процес розробки програмного забезпечення для розпізнавання рукописного тексту.

Предмет дослідження – програмні засоби для розпізнавання рукописного тексту.

Мета роботи – дослідження та розробка програмного забезпечення для розпізнавання рукописного тексту.

Рисунок В.2 – Слайд 2

ЗАВДАННЯ РОБОТИ

Для досягнення поставленої мети у кваліфікаційній роботі магістра необхідно розв'язати такі задачі:

- виконати аналіз предметної області та програмних засобів для розпізнавання рукописного тексту;
- розробити модель взаємодії користувача з програмним забезпеченням для розпізнавання рукописного тексту;
- здійснити проектування програмного забезпечення для розпізнавання рукописного тексту;
- створити програмне забезпечення для розпізнавання рукописного тексту;
- виконати тестування розробленого програмного забезпечення для розпізнавання рукописного тексту.

3

Рисунок В.3 – Слайд 3

ДОСЛІДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РОЗПІЗНАВАННЯ РУКОПИСНОГО ТЕКСТУ

Критерій порівняння	Transkribus	Mazec	Pen to Print
Висока точність розпізнавання	+	+-	+-
Багатомовність	+	+	+-
Можливість навчання власних моделей	+	+-	-
Зручність інтерфейсу	+-	+	+
Доступність на різних платформах	+-	+-	+-
Функціональність хмарного збереження	+	+-	+

4

Рисунок В.4 – Слайд 4

ВИБІР МОВИ ПРОГРАМУВАННЯ

Критерій порівняння мов програмування	Мова програмування		
	Python	Java	C++
Бібліотеки для машинного навчання та комп'ютерного зору	+	+—	+—
Простота синтаксису та швидкість розробки	+	+—	—
Активність та розмір спільноти у сфері штучного інтелекту	+	+—	+—
Зручність для створення ПЗ для розпізнавання рукописного тексту	+	—	—
Гнучкість у прототипуванні та тестуванні моделей	+	+—	—
Ефективність для роботи з великими наборами даних	+	+—	+—

5

Рисунок В.5 – Слайд 5

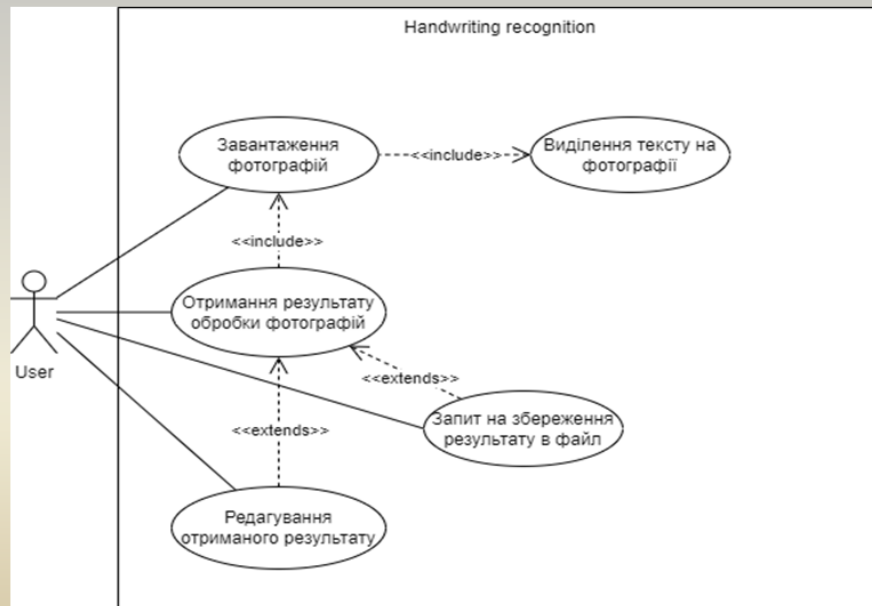
ВИБІР СЕРЕДОВИЩА РОЗРОБКИ

Критерій порівняння середовищ розробки	Середовища розробки		
	Visual Studio Code	PyCharm	Jupyter Notebook
Легкість налаштування та встановлення	+	+—	+
Гнучкість та розширюваність за рахунок плагінів	+	+—	—
Зручність роботи з великими проектами	+	+	—
Підтримка інструментів машинного навчання та штучного інтелекту	+	+	+—
Зручність налагодження коду	+	+	—

6

Рисунок В.6 – Слайд 6

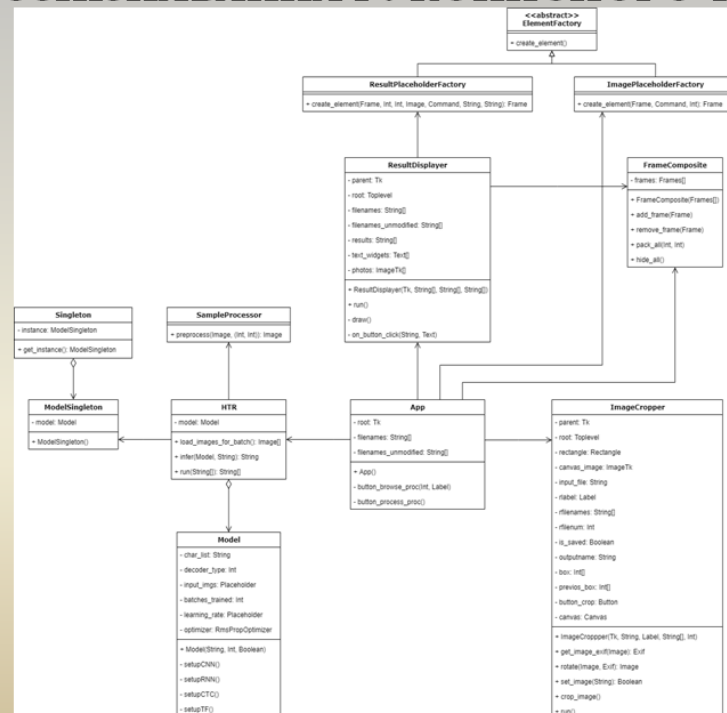
МОДЕЛЬ ВЗАЄМОДІЇ КОРИСТУВАЧА З ПЗ ДЛЯ РОЗПІЗНАВАННЯ РУКОПИСНОГО ТЕКСТУ



7

Рисунок В.7 – Слайд 7

ДІАГРАМА КЛАСІВ ПЗ ДЛЯ РОЗПІЗНАВАННЯ РУКОПИСНОГО ТЕКСТУ



8

Рисунок В.8 – Слайд 8

ФУНКЦІОНУВАННЯ ПЗ ДЛЯ РОЗПІЗНАВАННЯ РУКОПИСНОГО ТЕКСТУ

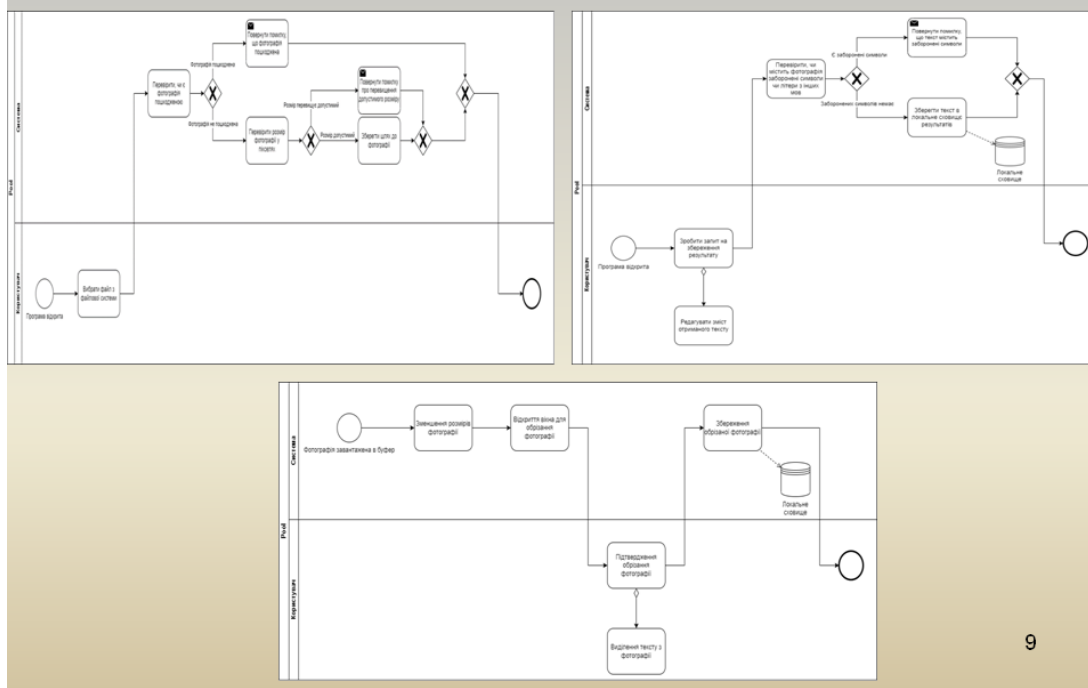


Рисунок В.9 – Слайд 9

НЕЙРОМЕРЕЖЕВА МОДЕЛЬ ДЛЯ РОЗПІЗНАВАННЯ РУКОПИСНОГО ТЕКСТУ

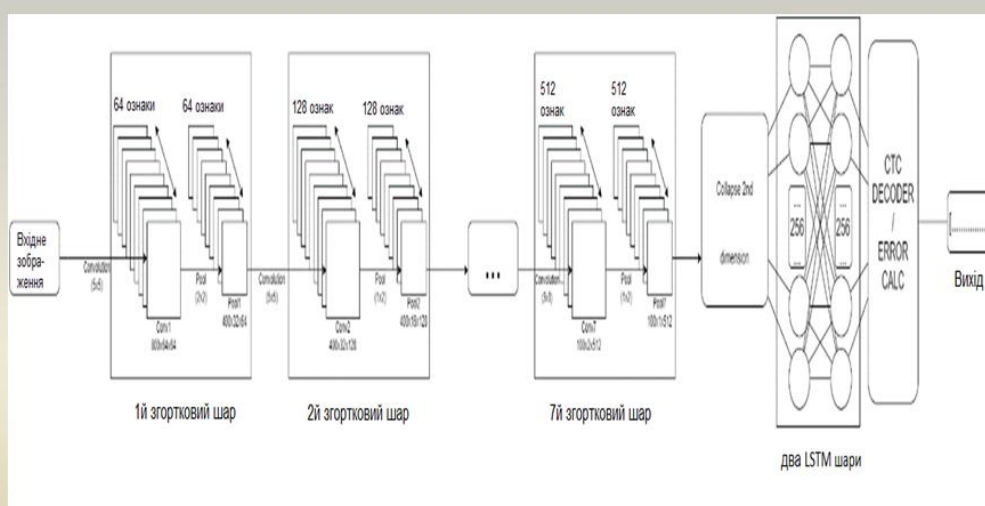


Рисунок В.10 – Слайд 10

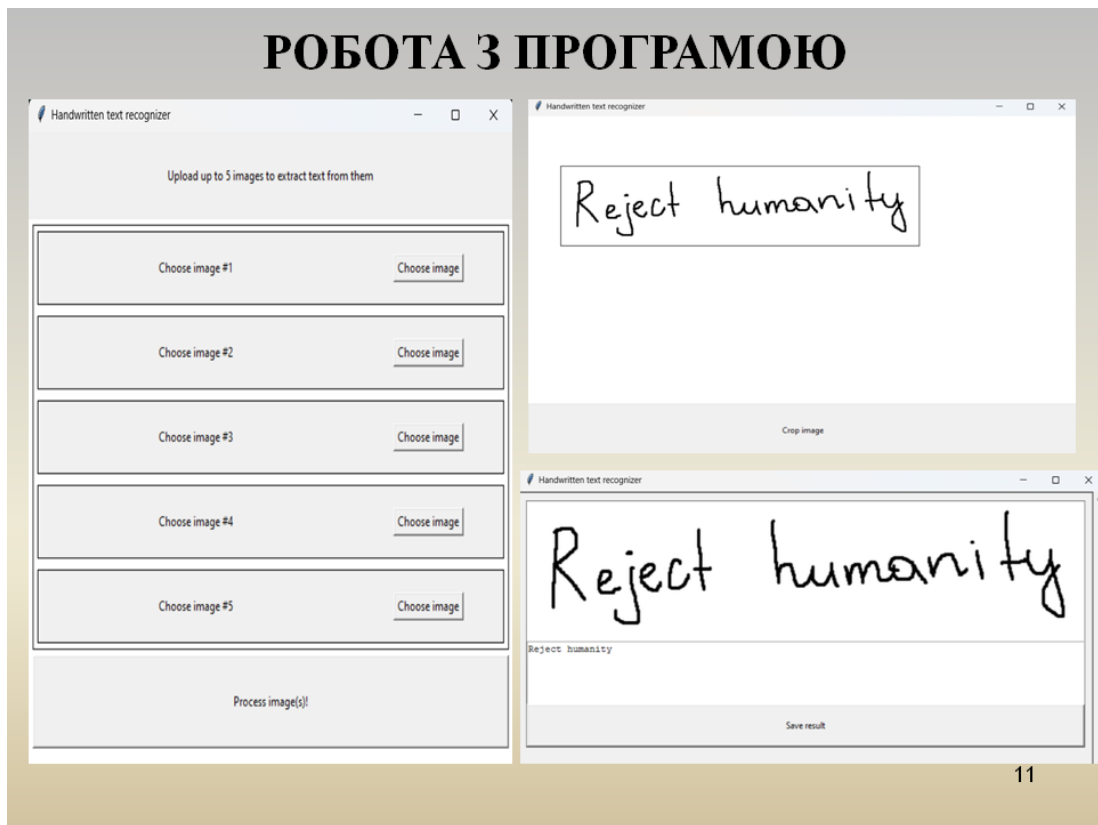


Рисунок В.11 – Слайд 11

Висновки

В ході виконання дипломної кваліфікаційної роботи магістра було проаналізовано та досліджено процес розробки програмного забезпечення для розпізнавання рукописного тексту.

Новизна роботи полягає у тому, що створено модель взаємодії користувача з програмним забезпеченням для розпізнавання рукописного тексту, яка подана у вигляді діаграми прецедентів, та дозволяє розроблювати програмне забезпечення для розпізнавання рукописного тексту.

Практична цінність роботи полягає у тому, що розроблено програмне забезпечення для розпізнавання рукописного тексту.

Виконано тестування розробленого програмного забезпечення. Результати тестування програмного забезпечення показали, що розроблена програма дозволяє забезпечити автоматизацію процесів, пов'язаних з підтримкою розпізнавання рукописного тексту.

12

Рисунок В.12 – Слайд 12