Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління
(повна назва)

Кафедра Комп'ютерних інтелектуальних технологій та систем
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)
Інтелектуальна система автоматизованого розпізнавання рукописного тексту
(тема)

Виконав:
здобувач 2025 року навчання,
групи КІУКІ-21-10
Олександр ШАТОХІН
(власне ім'я, прізвище)

Спеціальність 123 Комп'ютерна інженерія
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма
Комп'ютерна інженерія
(повна назва освітньої програми)

Керівник доц. каф. КІТС Олег ІЛЮНІН
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри _____ Олег РУДЕНКО
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління

Кафедра Комп'ютерних інтелектуальних технологій та систем

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 Комп'ютерна інженерія
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Комп'ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«____»_____ 20 ___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Шатохіну Олександру Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи Інтелектуальна система автоматизованого розпізнавання рукописного тексту

затверджена наказом університету від 21 травня 2025 р. № 399 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 20 червня 2025 р.

3. Вихідні дані до роботи

1. Набір даних для тренування нейронної мережі
2. Зображення рукописного тексту
3. Словник та World Beam Search
4. Параметри конфігурації системи
5. Використані технології: Python, PyTorch, SQLAlchemy Flask, ReactJS

4. Перелік питань, що потрібно опрацювати в роботі

1. Постановка задачі
2. Аналіз предметної області
3. Аналіз використаних технологій
4. Розробка та тренування моделі нейронної мережі
5. Реалізація серверної та клієнтської частини
6. Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) _____
_Слайд-презентація – 15 слайдів_____
_____
_____
_____
_____
_____
_____
_____
_____

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Затвердження дипломного проекту	26.05.2025	Виконано
2	Визначення мети, завдань та об'єкта розробки	27.05.2025-29.05.2025	Виконано
3	Створення архітектури інтелектуальної системи	29.05.2025-31.05.2025	Виконано
4	Збір та підготовка наборів даних	31.05.2025-02.06.2025	Виконано
5	Розробка архітектури програмного забезпечення	02.06.2025-05.06.2025	Виконано
6	Реалізація модуля попередньої обробки зображень	06.06.2025-08.06.2025	Виконано
7	Реалізація модуля розпізнавання рукописного тексту	08.06.2025-09.06.2025	Виконано
8	Навчання та оптимізація інтелектуальної системи	10.06.2025-13.06.2025	Виконано
9	Тестування та оцінка ефективності системи	14.06.2025-15.06.2025	Виконано
10	Оформлення пояснювальної записки	15.06.2025-19.06.2025	Виконано
11	Підготовка до захисту кваліфікаційної роботи	25.06.2025	Виконано

Дата видачі завдання _26__ ___травня____2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ _____доц. Ілюнін О.О._____
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 68 с., 26 рис., 2 дод., 18 джерел.

РОЗПІЗНАВАННЯ РУКОПИСНОГО ТЕКСТУ, НЕЙРОННА МЕРЕЖА, CRNN, CTC, КОМП’ЮТЕРНИЙ ЗІР, PYTHON, PYTORCH, FLASK, REACT, СЕГМЕНТАЦІЯ ТЕКСТУ, ОБРОБКА ЗОБРАЖЕНЬ

Метою кваліфікаційної роботи є розробка інтелектуальної системи, яка використовує методи комп'ютерного зору та глибокого навчання для автоматизованого перетворення зображень рукописного тексту у машиночитний формат.

Об'єктом дослідження є процеси обробки, аналізу та цифровізації інформації, що зберігається у рукописних документах.

Предметом дослідження є методи та моделі для розпізнавання рукописного тексту, зокрема архітектури на основі згорткових та рекурентних нейронних мереж з функцією втрат Connectionist Temporal Classification.

У роботі проведено аналіз та розробку end-to-end системи розпізнавання тексту. Для цього використовується архітектура CRNN, де згорткові шари виділяють візуальні ознаки з зображення, а двонапрямлена рекурентна мережа аналізує послідовність цих ознак для врахування контексту.

Розроблена система дозволяє автоматизувати процес оцифрування рукописів, що значно спрощує обробку документів, підвищує доступність інформації для пошуку та аналізу. Тестування на публічному датасеті IAM показало, що розроблена модель досягає конкурентоспроможної точності, що підтверджує ефективність обраного підходу.

ABSTRACT

The explanatory note of the qualification work: 68 pages, 26 figures, 2 appendices, 18 references.

HANDWRITTEN TEXT RECOGNITION, NEURAL NETWORK, CRNN, CTC, COMPUTER VISION, PYTHON, PYTORCH, FLASK, REACT, TEXT SEGMENTATION, IMAGE PROCESSING

The purpose of the qualification work is to develop an intelligent system that uses computer vision and deep learning methods for automated conversion of handwritten text images into machine-readable format.

The object of the research is the processes of processing, analysis and digitization of information stored in handwritten documents.

The subject of the research is methods and models for handwritten text recognition, in particular architectures based on convolutional and recurrent neural networks with the Connectionist Temporal Classification loss function.

The work analyzes and develops an end-to-end text recognition system. For this purpose, the CRNN architecture is used, where convolutional layers extract visual features from the image, and a bidirectional recurrent network analyzes the sequence of these features to take into account the context.

The developed system allows you to automate the process of digitizing manuscripts, which significantly simplifies document processing and increases the availability of information for search and analysis. Testing on the public IAM dataset showed that the developed model achieves competitive accuracy, confirming the effectiveness of the chosen approach.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API (Application Programming Interface) – Набір визначень і протоколів, що дозволяє програмним компонентам взаємодіяти між собою.

CER (Character Error Rate) – Метрика для оцінки точності систем розпізнавання тексту, що розраховується як відношення кількості неправильно розпізнаних символів до їх загальної кількості.

CNN (Convolutional Neural Network) – Згорткова нейронна мережа, особливо ефективна у задачах комп'ютерного зору.

CRNN (Convolutional Recurrent Neural Network) – Згортково-рекурентна нейронна мережа.

CSV (Comma-Separated Values) – Текстовий формат, призначений для представлення табличних даних, де кожний рядок відповідає одному рядку таблиці, а значення в рядку розділяються комами.

CTC (Connectionist Temporal Classification) – Функція втрат, що використовується для навчання нейронних мереж у задачах розпізнавання послідовностей, де відсутнє точне вирівнювання між вхідними та вихідними даними.

DOM (Document Object Model) – Програмний інтерфейс для HTML та XML документів, що представляє сторінку у вигляді структури об'єктів, дозволяючи динамічно змінювати її вміст.

GPU (Graphics Processing Unit) – Графічний процесор.

HMM (Hidden Markov Model) – Статистична модель, що дозволяє описувати системи, які переходять між різними станами з певними ймовірностями.

HTR (Handwritten Text Recognition) – Технологія, що дозволяє автоматично перетворювати зображення рукописного тексту на машиночитний формат.

IAM (IAM Handwriting Database) – Широко використовуваний у наукових дослідженнях набір даних, що містить зразки рукописного англійського тексту від великої кількості авторів.

JSON (JavaScript Object Notation) – Текстовий формат обміну даними, що базується на синтаксисі JavaScript і є легкочитним як для людей, так і для машин.

LSTM (Long Short-Term Memory) – Тип рекурентних нейронних мереж, здатних запам'ятовувати інформацію протягом тривалих періодів часу, що робить їх ефективними для аналізу послідовностей.

OCR (Optical Character Recognition) – Технологія, що дозволяє розпізнавати друкований або рукописний текст на зображеннях і перетворювати його в електронний формат.

PDF (Portable Document Format) – Формат файлів, розроблений для представлення документів незалежно від програмного забезпечення, апаратних засобів та операційних систем.

PNG (Portable Network Graphics) – Растровий формат зберігання графічної інформації, що використовує стиснення без втрат.

RNN (Recurrent Neural Network) – Рекурентна нейронна мережа; клас нейронних мереж, де зв'язки між вузлами утворюють спрямований граф вздовж часової послідовності, що дозволяє їм обробляти дані з пам'яттю про попередні події.

WER (Word Error Rate) – Метрика, що оцінює якість розпізнавання шляхом підрахунку відсотка слів, у яких допущено хоча б одну помилку.

ВСТУП

У сучасну цифрову епоху автоматизація обробки інформації набуває все більшого значення. Велика кількість цінних даних досі існує у вигляді рукописних документів – історичних архівів, нотаток, анкет та інших матеріалів. Розпізнавання рукописного тексту є технологією, що дозволяє перетворити зображення з рукописом у цифровий текст. Це відкриває можливості для збереження культурної спадщини, спрощення обробки документів і підвищення доступності інформації для пошуку та аналізу. Застосування таких систем охоплює різні сфери – від оцифрування історичних рукописів і нотаріальних книг до автоматичного введення даних з анкет або перевірки тестів у сфері освіти.

Розпізнавання рукописів є значно складнішою задачею, ніж класичне оптичне розпізнавання надрукованого тексту . Якщо для машинного читання друкованих символів існують відлагоджені рішення з високою точністю, то рукописний текст характеризується величезною варіативністю почерків, стилів написання, розмірів і нахилів літер. Навіть один і той самий автор може писати по-різному залежно від обставин. На відміну від друкованих шрифтів, кожен рукопис унікальний, що створює додаткові труднощі для автоматичного розпізнавання. Крім того, в рукописному тексті літери часто з’єднуються між собою, можуть бути присутні артефакти (плями, розмиви), а скановані зображення нерідко містять шуми та перекоси. Усе це обумовлює необхідність розвитку спеціалізованих методів і алгоритмів для HTR.

Отже, враховуючи складність завдання та значущість рукописної інформації, дослідження й розробка ефективних методів її автоматичного розпізнавання залишаються пріоритетним напрямком у наші часи. Подальша робота над вдосконаленням алгоритмів HTR, підвищенням їх точності та адаптивності до різних стилів письма є важливим кроком на шляху до більш повної цифровізації інформаційного простору.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Еволюція та сучасний стан технологій розпізнавання тексту

Проблема автоматичного читання тексту має давню історію в галузі комп’ютерного зору та штучного інтелекту. Перші системи оптичного розпізнавання символів з’явилися ще в середині XX століття і успішно використовувалися для читання друкованих текстів (наприклад, для автоматичного сортування пошти). Однак для рукописного тексту прогрес тривалий час був повільнішим через набагато більшу варіативність вхідних даних. Ранні підходи до HTR базувалися на жорстких правилах і шаблонах: система порівнювала зображення символів із наперед заданими еталонами. Такі рішення могли працювати лише для обмеженого набору почерків і вимагали налаштування під кожного користувача.

В 1990-х роках ситуація почала змінюватися з появою статистичних моделей та перших успішних нейронних мереж. Розвиток алгоритмів машинного навчання дав змогу навчати системи розпізнавання на основі прикладів, що значно підвищило їх гнучкість. Велика увага приділялася використанню прихованих марковських моделей та нейронних мереж для задач розпізнавання рукописних слів. На рубежі 2000-х було створено кілька великих баз даних рукописного тексту (наприклад, IAM та інші), які стимулювали дослідників до розробки та порівняння нових підходів [1].

Сьогодні сучасний стан HTR визначається домінуванням глибокого навчання. Завдяки потужності глибоких нейронних мереж, вдалося досягти суттєвого прогресу – системи здатні розпізнавати рукописний текст цілими рядками і навіть сторінками. Застосування згорткових нейронних мереж для виділення особливостей зображення та рекурентних мереж для моделювання послідовностей дозволило будувати ефективні end-to-end моделі розпізнавання. Наприклад, використання двонаправлених LSTM у комбінації

з CTC-ланцюгами стало проривом, що на довгі роки визначив розвиток HTR. Нині найкращі результати демонструють архітектури на основі глибоких CNN і трансформерів, здатні враховувати контекст на рівні всього рядка або абзацу [2]. Важливо відзначити, що завдання HTR більше не обмежується лише розпізнаванням окремих слів – сучасні системи спрямовані на цілісне читання рукописних документів з урахуванням структурованого розташування тексту. Тим не менш, навіть найсучасніші моделі поки що поступаються людині у точності, особливо на складних або незвичних почерках

1.2 Основні труднощі розпізнавання рукописних текстів

Розробка системи HTR пов’язана з рядом специфічних викликів. Перш за все, це надзвичайна різноманітність почерків. Кожна людина пише у своєму власному стилі: від акуратного каліграфічного до швидкого та недбалого письма. Літери можуть мати різну форму, нахил, розмір; слова – різний інтервал між символами; лінії тексту – нерівномірний горизонтальний нахил тощо . Унаслідок цього одні й ті самі символи можуть виглядати дуже по-різному, і алгоритм, що добре розпізнає текст однієї особи, може помилятися на тексті іншої.

Другою проблемою є накладання і злиття символів. У рукописному тексті, особливо курсивному, сусідні літери часто з’єднуються, утворюючи безперервний штрих. Межі між символами розмиті, що ускладнює сегментацію – поділ зображення на окремі символи чи слова. Наприклад, літери «м» і «ш» у деяких почерках відрізняти важко, так само як послідовності на кшталт «rn» та «m» можуть виглядати схоже.

Третій виклик – якість зображення. Рукописні тексти часто доступні у вигляді відсканованих копій або фотографій, які можуть мати шуми: плями від чорнил, розмиття фокусу, нерівномірне освітлення, артефакти стиснення тощо. Такі шуми спотворюють зображення символів. Також, фізичні

документи можуть містити помарки, закреслення, нестандартні розміщення тексту (наприклад, надрядкові примітки), що додатково ускладнює аналіз.

Ще одна проблема – перекіс і вирівнювання тексту. При скануванні сторінок часто трапляється, що рядки тексту нахилені під кутом до горизонталі. Для успішного розпізнавання потрібне вирівнювання – обертання зображення на невеликий кут, щоб рядки стали горизонтальними. Аналогічно, треба враховувати можливі перспективні викривлення, якщо фото зроблено під кутом (рисунок 1.1).

Рисунок 1.1 – Приклади складнощів рукописного тексту

На відміну від друкованого тексту, де зазвичай присутня перевірка орфографії, рукописний текст може містити нестандартні скорочення, авторські стилі письма, помилки. Система розпізнавання повинна не лише ідентифікувати окремі символи, а й формувати з них слова, співставні із справжньою мовою. Відсутність контексту або словника може призводити до того, що навіть правильно розпізнані літери утворюють некоректне слово. Отже, для підвищення якості розпізнавання бажано залучати мовні моделі – статистичні або нейронні моделі мови, які допомагають виправити очевидні помилки на основі частоти вживання слів.

1.3 Основні підходи до розпізнавання рукописного тексту

Увесь спектр рішень, які пропонуються для читання рукописів, можна умовно розділити на три покоління: від ранніх алгоритмів зі строгими шаблонами, через статистичні моделі з імовірнісними правилами до сучасних нейромереж, що навчаються «з нуля» на великих масивах даних.

1.3.1 Традиційні алгоритмічні методи на основі шаблонів та ознак

Перші системи HTR здебільшого використовували алгоритмічні методи без навчання, покладаючись на заздалегідь визначені правила. Класичний приклад – шаблонне співставлення, коли зображення символу порівнюється зі зразками еталонних літер. Наприклад, для кожної букви алфавіту могли бути збережені кілька шаблонних зображень, а для розпізнавання проводився пошук найбільш подібного шаблону (за метрикою кореляції або мінімальної різниці пікселів). Такий підхід працює швидко та добре у разі фіксованих шрифтів, але майже непридатний для рукописів через різноманітність почерків. Необхідно було б зберігати надто багато шаблонів під різні варіанти написання кожної літери, і все одно система залишалась би нечутливою до навіть невеликих відхилень.

Інший напрям ранніх підходів – виділення характерних геометричних ознак символів з подальшою класифікацією. В якості ознак могли використовуватися, наприклад, кількість перетинів рядкової лінії, наявність замкнутих областей у літері, співвідношення висоти і ширини, напрямки штрихів тощо. Виділені ознаки формували вектор, який потім порівнювався з еталонними зразками (класифікація на основі відстаней або евристичних правил). Такі системи могли частково враховувати варіативність почерку, але вимагали ручного конструювання ознак та часто давали збої на нестандартних випадках.

Деякого поширення набули методи структурного розпізнавання, де

кожен символ описувався набором примітивів (ліній, дуг, крапок перетину). Розпізнавання полягало у виявленні цих примітивів на зображенні та перевірці, чи утворюють вони відому літеру. Наприклад, букву «А» можна описати як поєднання двох діагональних і однієї горизонтальної ліній, а «М» – як послідовність зигзагоподібних штрихів. Але такі підходи також страждали від чутливості до варіацій – якщо літера написана нетипово, алгоритм може не впізнати потрібні примітиви.

Загалом, традиційні методи мають обмежену стійкість до розмаїтості рукописного почерку. Вони можуть працювати для окремих специфічних задач (наприклад, розпізнавання строгих бланків або цифр на банківських чеках), але в загальному випадку поступаються методам машинного навчання, які будуть розглянуті далі.

1.3.2 Статистичні моделі та приховані марковські моделі

З появою потужніших обчислювальних ресурсів і накопиченням великих обсягів даних рукописів, у 1990-х роках дослідники почали активно застосовувати статистичні моделі для задачі розпізнавання рукописного тексту. Одним з найбільш успішних підходів того періоду стали приховані марковські моделі. Вони добре підходять для моделювання послідовностей, тому природно застосовувалися до послідовностей ознак, отриманих із зображення тексту [2].

У типовому підході на основі HMM кожне слово (або навіть кожна літера) моделюється окремою марковською моделлю – послідовністю станів, які генерують вихідні символи з певними ймовірностями. Щоб розпізнати слово на зображенні, потрібно було спочатку перетворити зображення на послідовність ознак. Зазвичай для цього використовували метод скануючого вікна: вузьке вертикальне «вікно» рухалося уздовж зображення рядка тексту і будувало ознаки для кожної позиції (наприклад, вектор з кількостей чорних пікселів у різних горизонтальних секціях). Таким чином, рукописне слово

перетворювалось на послідовність векторів ознак, які подавалися на вхід HMM-моделі. Далі методами теорії марковських процесів обчислювалася найбільш ймовірна послідовність станів, тобто визначалося, яку послідовність символів ця модель генерує з максимальною правдоподібністю для даного зображення. Іншими словами – відшуковувалось розпізнане слово.

Перевагою HMM було те, що вони не потребували явної сегментації на окремі літери – модель самостійно «розтягувала» послідовність станів під довжину вхідної послідовності ознак, дозволяючи таким чином працювати з цілим словом або рядком. Це зняло одну з головних проблем – необхідність точно знаходити межі кожної букви.

Але HMM мали і суттєві недоліки. По-перше, марковські моделі мають властивість «memorylessness» – ймовірність переходу визначається лише поточним станом, без довготривалої пам’яті про попередні символи. Це ускладнює моделювання довгих залежностей у рукописі (наприклад, повторюваних елементів почерку). По-друге, якість розпізнавання сильно залежала від вибору ознак – ручне конструювання ознак не завжди давало змогу повно охопити всі нюанси написання. Так, результати чисто HMM-підходів хоч і були кращими за шаблонні методи, проте залишалися далекими від бажаних.

Щоб підвищити точність, дослідники впроваджували гібридні системи, поєднуючи HMM з іншими підходами. Наприклад, успішними були комбінації HMM з гаусовими сумішами (HMMGMM) для більш гнучкого моделювання розподілу ознак; нейронними мережами для оцінки ймовірностей переходів або вихідних символів (утворюючи архітектури типу HMM+MLP або HMM+CNN); рекурентними нейронними мережами, що додавали пам’ять (наприклад, поєднання HMM з LSTM-мережею).

Такі гібридні моделі показували значно кращі результати, ніж класичні HMM. В деяких випадках вдавалося досягти прийнятної точності розпізнавання слів на обмежених словниках. Проте загальний тренд у галузі

поступово схилявся від HMM до повністю нейронних рішень, про які мова піде далі.

1.3.3 Сучасні нейронні мережі глибокого навчання

З середини 2000-х років із розвитком глибокого навчання кардинально змінився підхід до розпізнавання рукописного тексту. Замість ручного проектування ознак і використання моделей з лінійними допущеннями, дослідники почали будувати нейронні мережі, які самі навчаються оптимальним ознакам і правилам розпізнавання, отримуючи дані у сирому вигляді, як пікселі зображення.

Одним з піонерів у цій сфері став Алекс Грейвс, який запропонував застосувати рекурентні нейронні мережі з довгою короткочасною пам’яттю для послідовного «читання» рукопису зображення рядка тексту без попередньої сегментації на символи. Ключовим компонентом цієї архітектури був шар CTC (Connectionist Temporal Classification) – спеціальна функція втрат, що дозволяє тренувати мережу порівнювати всю послідовність виходів із цільовим текстом, автоматично співставляючи вихідні символи з позиціями у слові [4]. Це усуває потребу у ручній розмітці меж кожного символу на тренувальних даних – мережа сама навчиться розтягувати чи стискати вихід під довжину слова.

Типова сучасна нейронна архітектура для HTR складається з двох основних компонентів: згорткового фронтенду та рекурентного бекенду. На першому етапі згорткові нейронні мережі виділяють візуальні ознаки зображення – різноманітні фільтри виявляють контури, форми літер, особливості почерку. Глибокий CNN здатний перетворити зображення рядка на послідовність векторів-ознаків (фактично, що бачимо у кожному фрагменті зображення). Далі ці послідовності ознак обробляються рекурентною мережею або самонастроюваною трансформерною моделлю, яка аналізує послідовність у часі і генерує вихідну послідовність символів.

Важливо, що рекурентна мережа може запам’ятовувати попередній контекст, тому, на відміну від HMM, враховує довготривалі залежності – наприклад, розтягнуті елементи літер або повторення. На фінальному етапі додається повнозв’язний шар, який перетворює виходи RNN у ймовірності конкретних символів на кожній позиції. Мережа навчається шляхом мінімізації CTC-втрат, що змушує її давати найбільшу сумарну правдоподібність саме правильній послідовності символів.

Подібні архітектури, відомі як CRNN, наразі є стандартом для HTR. Вони успішно використовуються у більшості переможців змагань з розпізнавання рукопису і демонструють високі показники точності на різних наборах даних.

Відзначимо, що останніми роками набирають популярності і моделі без рекурентних компонентів – засновані на трансформерах з механізмом уваги. Такі моделі, запозичені з сфер розпізнавання мови та машинного перекладу, можуть прямо перетворювати зображення на текст, навчаясь відповідності фрагментів зображення символам у виході. В деяких випадках трансформерні HTR-системи навіть перевершують RNN-аналоги, особливо на довгих текстах, оскільки краще враховують контекст [5].

Можна зазначити, що сучасний етап розвитку HTR характеризується переходом до комплексних нейронних рішень, які мінімізують роль ручної обробки. Ці рішення вже зараз забезпечують досить високу якість розпізнавання друкованих і рукописних текстів, що наближається до людини, але все ще залишається досконалюватися, особливо на нетипових даних та в складних умовах.

1.4 Існуючі системи та їх обмеження

На сьогодні розроблено чимало практичних систем оптичного розпізнавання тексту. Багато з них орієнтовані переважно на друкований текст (наприклад, відомі програми ABBYY FineReader, Adobe OCR, Tesseract

тощо) і демонструють відмінні результати на відсканованих друкованих документах – точність впізнавання символів часто перевищує 95%. Проте стосовно рукописного тексту універсальні OCR-движки справляються значно гірше. Наприклад, популярний відкритий рушій Tesseract OCR, не має у своєму стандартному пакеті готової моделі для рукописів [6]. Дослідники відзначають, що підтримка рукописного тексту в Tesseract досі залишається обмеженою. Не існує попередньо натренованих даних для рукописів, а самостійне навчання LSTM-моделі Tesseract на рукопис потребує значних зусиль і не гарантує високої точності. Фактично, при використанні «з коробки» Tesseract може розпізнавати лише окремі дуже прості рукописні символи і вимагає інтенсивної передобробки зображень, щоб хоч якось прочитати рукописний текст. Покращення якості зображення (бінаризація, фільтрація шумів) є необхідним кроком перед передачею в OCR-рушій (рисунок 1.2).

Рисунок 1.2 – Приклад бінаризації зображення

Альтернативою відкритим рішенням є комерційні сервіси та спеціалізовані API для розпізнавання тексту, які надають великі компанії. Наприклад, Google Cloud Vision та Microsoft Azure OCR підтримують режим розпізнавання рукописного тексту. Ці сервіси використовують власні глибокі нейронні моделі, натреновані на величезних вибірках, і можуть досягати досить високої точності – за окремими звітами, до 90% і більше правильно розпізнаних символів на зрозумілих рукописах [7]. Однак їх використання пов’язане з платою за обробку, а також із вимогою передавати дані на зовнішній сервер, що не завжди прийнятно з точки зору безпеки та конфіденційності (наприклад, для медичних документів або приватних записів).

Існують також спеціалізовані системи, розроблені для окремих задач: розпізнавання поштових індексів і адрес, читання цифр на банківських чеках, системи для автоматичного вводу рукописних анкет. Такі системи можуть комбінувати декілька підходів – наприклад, спочатку нейронна мережа вгадує кілька найімовірніших варіантів символу, а потім зі словника або за контекстом обирається найбільш схожий варіант слова. Однак вони зазвичай обмежені конкретною предметною областю і не є загальним рішенням для будь-якого рукописного тексту.

До основних обмежень існуючих HTR-систем можна віднести:

− Чутливість до почерку: модель, навчена на одних почерках, може давати збої на інших (особливо це стосується систем без глибокого навчання). Нерідко на практиці потрібно донавчати або адаптувати систему під конкретного користувача.

− Межі точності: навіть найкращі сучасні моделі припускаються помилок. Наприклад, для популярного англомовного набору IAM точність розпізнавання слів у провідних моделей становить близько 80–90%, що означає, що на кожні 10 слів 1–2 слова можуть бути прочитані неправильно. Цього недостатньо для деяких критичних застосувань без додаткової перевірки людиною.

− Вимоги до ресурсів: глибокі нейронні мережі для HTR є великими за розміром і потребують значних обчислювальних ресурсів (GPU) для навчання і навіть для швидкої роботи. Вбудовувати їх у мобільні пристрої або мікроконтролери поки що складно, тому часто потрібне підключення до хмарного сервісу.

− Мова і алфавіт: більшість комерційних рішень найкраще працюють для англійської та кількох інших мов. Підтримка української мови або, наприклад, ієрогліфічних систем письма може бути обмеженою або відсутньою. Необхідність збирання великого датасету для кожної нової мови ускладнює розширення універсальних систем.

− Формат документа: окрім власне тексту, документи можуть містити таблиці, формули, незвичайні шрифти чи символи. Сучасні HTR-системи, як правило, орієнтовані лише на безперервний рукописний текст. Розпізнавання змішаного вмісту (наприклад, текст та формули) все ще є відкритою проблемою.

Загалом, хоча нинішні системи здатні автоматично читати рукописи значно краще, ніж це було десятиліття тому, перед ними стоїть ще багато невирішених завдань. Розробники змушені шукати компроміс між універсальністю рішення та його точністю і потребами в ресурсах. У цьому контексті дослідження в галузі HTR залишаються актуальними, а вдосконалення існуючих алгоритмів – затребуваним напрямом.

1.5 Актуальні виклики та напрямки розвитку HTR

Беручи до уваги розглянуті проблеми, можна виділити кілька ключових викликів, що наразі стоять перед розпізнаванням рукописного тексту, та відповідні напрями активних досліджень.

1.5.1 Обмежені дані та варіативність почерків

Для навчання високоточної нейронної моделі потрібні великі обсяги розмічених даних – зображень рукописів з відомим текстом. Зібрати такі дані складно, особливо для мови з обмеженою кількістю носіїв або для специфічних типів документів (наприклад, середньовічні рукописи). Бази даних на кшталт IAM містять трохи більше ста тисяч слів, чого недостатньо для повного охоплення всіх стилів письма. До того ж, у них представлено лише певну вибірку авторів. На практиці ж почерки мільйонів людей можуть відрізнятися куди сильніше. Це створює ризик, що модель не узагальнить на незнайомий почерк.

Для подолання цієї проблеми застосовують такі підходи:

− Аугментація даних – штучне збільшення тренувального набору шляхом додавання модифікованих копій наявних зразків (додавання шуму, зміна масштабу, деформація літер тощо). Такий підхід використано і в нашій реалізації при підготовці даних, що допомагає моделі стати стійкішою до дрібних випадкових варіацій (рисунок 1.3).

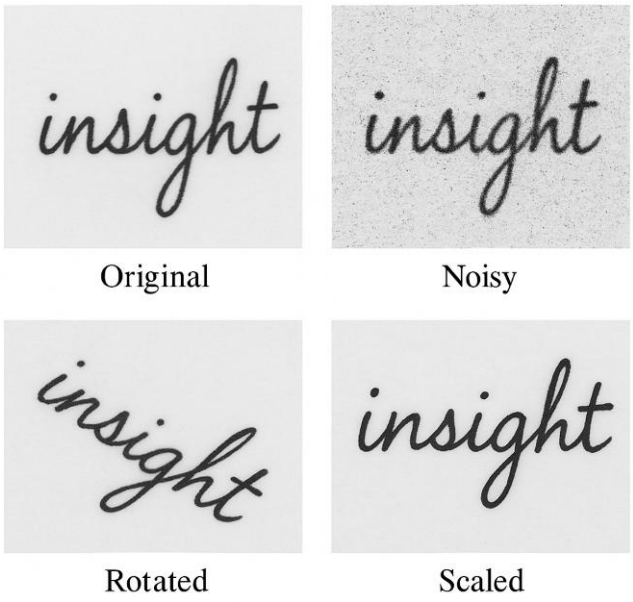

Рисунок 1.3 – Приклад аугментації даних

− Генерація синтетичних даних. Один зі шляхів – створювати згенеровані зображення тексту різними шрифтами, що імітують почерк, або навіть використовувати спеціальні генеративні моделі для синтезу псевдо-рукописів. Синтетичні дані можуть значно збільшити різноманітність стилів письма у тренувальному наборі.

− Адаптація моделі до нового почерку. Якщо система планується для конкретного користувача або вузької групи, можливий підхід адаптувати загальну модель під їхній почерк. Для цього користувач пише невеликий текст як зразок, модель донавчається на цьому, після чого точність для саме цього почерку істотно зростає. Безвчительне або слабко-вчительне навчання. Актуальний напрям – навчання моделей на нерозмічених даних (self-supervised learning), коли модель вчиться витягувати структуру рукопису без прямих пар «зображення-текст». Це може дозволити задіяти великі корпуси сканів без ручної розмітки, зменшуючи залежність від вручну підготовлених датасетів.

1.5.2 Розпізнавання документів і контекстне післяопрацювання

Більшість сучасних систем, у тому числі реалізованих у рамках цього проекту, виконують розпізнавання на рівні окремо взятих рядків або слів. Наприклад, у історичному рукописному листі текст може бути написаний нерівними колонками, додатково містити поля з примітками. Завдання сегментації сторінки на логічні блоки тексту є нетривіальним і досі розвивається як окрема підзадача [8]. Інший момент – контекстна корекція результатів розпізнавання. Навіть якщо нейромережа помилилась у кількох літерах, але виходить слово, не схоже на жодне відоме слово мови, системі можна «підказати» виправлення, використовуючи словники або мовну модель Впровадження такого післяопрацювання істотно підвищує якість кінцевого результату: виправляє окремі помилки і надає узгодженості тексту.

2 АНАЛІЗ ВИКОРИСТОВУВАНИХ ТЕХНОЛОГІЙ

2.1 Мова програмування Python і середовище розробки

Основним засобом реалізації програмної частини стала мова програмування Python. Вибір зумовлений тим, що Python є де-факто стандартом у сфері машинного навчання та обробки зображень завдяки великій кількості доступних бібліотек і зручності швидкої розробки прототипів. Зокрема, у Python представлені високорівневі бібліотеки для роботи з нейронними мережами, такі як TensorFlow, Keras, PyTorch та інші, а також багаті засоби для наукових обчислень і обробки даних. Інтерактивні інструменти, наприклад Jupyter Notebook, дозволяють поєднувати виконання коду з візуалізацією результатів, що дуже зручно при експериментальній розробці алгоритмів розпізнавання.

Розробка проведена у середовищі Jupyter Notebook, яке надало можливість поетапно виконувати фрагменти коду, одразу спостерігаючи вивід – зображення, графіки, статистику [9]. Це значно спрощує налагодження процесу обробки зображень та навчання моделі. Наприклад, на етапі передобробки ми інтерактивно переглядали результати бінаризації та сегментації для кількох сторінок, коригуючи параметри до отримання прийнятного результату.

Варто відзначити, що для ефективного навчання глибокої нейронної мережі використовувався доступ до графічного процесора. Навчання моделі проводилось у середовищі, оснащеному відеокартою NVIDIA, що дозволило суттєво прискорити обчислення. Багатоопераційні задачі, такі як обчислення градієнтів у нейромережі, виконуються на GPU в десятки разів швидше, ніж на CPU [10]. У середовищі Jupyter була підключена підтримка GPU через відповідні драйвери та бібліотеки.

Для керування залежностями та версіями бібліотек використовувався

менеджер пакетів pip. З його допомогою було встановлено необхідні пакети: OpenCV для обробки зображень, pytorch та pytorch-lightning для створення і навчання моделі, albumentations для аугментації даних та інші допоміжні бібліотеки, такі як matplotlib для візуалізації, pandas для роботи з таблицями анотацій тощо [11].

Поєднання мови Python з інтерактивним середовищем Jupyter Notebook створило гнучкий інструментарій, який дозволив зосередитись на логіці алгоритмів розпізнавання, а не на низькорівневих деталях реалізації. Використання готових бібліотек значно прискорило розробку: було легко завантажувати та опрацьовувати зображення, будувати нейронну мережу високого рівня складності, оцінювати якість розпізнавання – все це у кількох рядках зрозумілого коду.

2.2 Датасет та дані для навчання

Для навчання та перевірки працездатності моделі був обраний відкритий датасет IAM Handwriting Database – один з найбільш відомих і широко використовуваних наборів даних у сфері HTR . Даний датасет містить зразки рукописного тексту англійською мовою, зібрані в умовах, близьких до реального письма. Його особливості:

− 657 авторів – тексти написані великою кількістю різних людей, що забезпечує різноманітність почерків.

− 1539 сторінок рукописного тексту. Кожна сторінка – це заповнений від руки бланк зі зв’язним англійським текстом.

− 5685 речень, 13353 рядків та 115320 окремих слів, вирізаних зі сторінок та розмічених відповідними текстовими транскрипціями. Датасет містить розмітку на рівні речення, рядка і слова – тобто для кожного такого фрагмента відома послідовність символів, яка на ньому зображена.

− Алфавіт: датасет охоплює 79 унікальних символів, включно з літерами, цифрами, розділовими знаками та пропуском. Це означає, що

модель повинна підтримувати не лише літери англійського алфавіту, а й цифри та деякі спеціальні символи, якщо вони трапляються. Зображення надані у відтінках сірого (256 градацій), з роздільною здатністю 300 точок на дюйм. Якість сканів досить висока, але присутні незначні шуми, іноді є плями або артефакти фону [12].

Датасет IAM був обраний через його публічну доступність та велику популярність у науковій спільноті – результати, отримані на ньому, легко порівнювати з іншими підходами. Крім того, він зручний тим, що вже містить готові нарізки слів та їх транскрипції у текстовому файлі words.txt (рисунок 2.1).

Рисунок 2.1 – Приклади слів з датасету

Однак, для демонстрації повного циклу обробки, було реалізовано і процедуру сегментації сирого зображення сторінки на рядки та слова. Втім, для навчання моделі ми скористалися вже готовими нарізками слів із датасету, оскільки вони точніше розмічені та охоплюють більший обсяг даних, ніж можна було б обробити вручну у рамках проекту.

Розподіл даних на тренувальну і тестову частини здійснювався відповідно до стандартної методики, запропонованої авторами датасету. Зазвичай IAM має наперед визначений список сторінок для тренування, валідації та тесту. У нашому випадку було обрано близько 90% наявних авторів для тренування, а ~10% – для валідації моделі. Такий підхід гарантує,

що у валідованих даних зустрічаються лише почерки, яких не було в навчанні – це дозволяє оцінити здатність моделі до узагальнення на нових користувачах.

На кінець цього етапу ми мали підготовлений CSV-файл із переліком усіх вибраних зображень слів та відповідних текстів. Після фільтрації були вилучені кілька пошкоджених файлів. Також було сформовано словник символів розміром 80 символів – це всі різні символи, що зустрічаються в транскрипціях, плюс спеціальний символ «пропуск» для CTC-алгоритму. Цей словник потрібен моделі для відображення виходу нейронної мережі (числових індексів) у власне символи при декодуванні.

2.3 Інструменти обробки зображень та бібліотека OpenCV

Важливою частиною системи є передобробка зображень перед подачею їх в нейронну мережу. Для цієї мети використовувалася бібліотека OpenCV – відкрита крос-платформна бібліотека функцій комп’ютерного зору. OpenCV надає широкий набір інструментів для роботи з зображеннями: завантаження, зміна розміру, перетворення в відтінки сірого, згладжування, порогова обробка, виділення контурів тощо [13]. У нашій задачі з OpenCV були реалізовані ключові етапи обробки:

− Конвертація до рівнів сірого – кольорові зображення переводяться у градації сірого для спрощення подальшого аналізу. У випадку IAM-датасету сторінки вже надано у відтінках сірого.

− Фільтрація шуму – для згладжування дрібних артефактів використовувався гаусівський фільтр. Це допомагає прибрати окремі шуми і полегшити побудову чіткої бінарної маски тексту.

− Бінаризація – перетворення зображення у чорнобілий. Було застосовано глобальний поріг з інверсією кольорів. Тобто, алгоритм автоматично підбирав поріг яскравості, за яким пікселі розділяються на «чорні» (текст) та «білі» (тло) таким чином, щоб мінімізувати

внутрішньокласову дисперсію. Інверсія була потрібна тому, що зручніше працювати з текстом, представленим значенням 1 на фоні 0 (особливо це важливо для нейронної мережі, яка буде подаватися на вхід – переважно текст зробили білим, а фон чорним). Бінаризація є критично важливою для подальшої сегментації, адже після неї зображення набуває вигляду чітких чорних областей на однорідному білому фоні [14].

– Сегментація рядків тексту – завдання виділити окремі горизонтальні рядки на сторінці. Ми використали метод проєкцій: рахується горизонтальна проєкція білих пікселів – тобто для кожного y-рядка зображення рахується кількість чорних точок. Ця проєкція є одновимірним масивом, де сегменти з високими значеннями відповідають зонам з текстом, а близькі до нуля – проміжкам між рядками. Задавши поріг (ми брали ~10% від максимального значення), можна визначити, де проєкція достатньо велика, щоб вважати цю область рядком. Проходячи по проєкції, ми фіксуємо початок рядка і кінець рядка. Так збираються координати всіх рядків. Було додано також мінімальне обмеження на висоту рядка, щоб відкидати випадкові дрібні сегменти. У результаті для тестової сторінки метод знайшов, наприклад, 13 рядків [15]. Нижче показано, як червоними лініями позначено межі виявлених рядків на зображенні сторінки (рисунок 2.2).

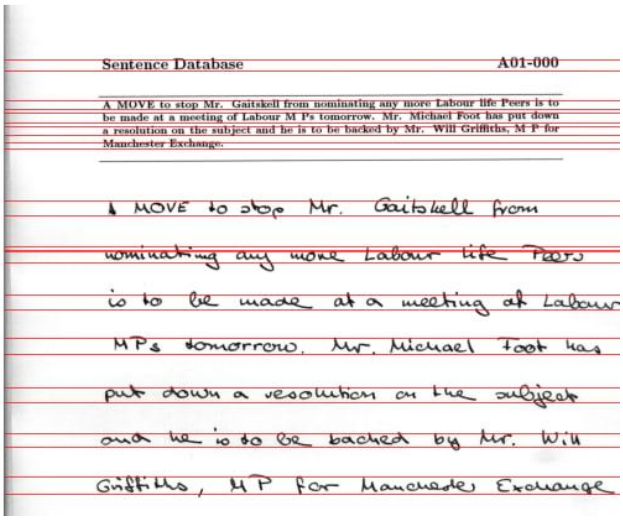

Рисунок 2.2 – Приклад сегментації тексу на рядки

За допомогою OpenCV отримані фрагменти зображень слів були збережені на диск у форматі PNG для подальшого використання у навчанні нейронної мережі. Таким чином, OpenCV відіграла роль «очей» системи – вона перетворила сирі візуальні дані в структуровану форму, придатну для аналізу.

2.4 Фреймворк глибокого навчання Pytorch та Lightning

Серцем системи розпізнавання є глибока нейронна мережа. Для її побудови та навчання ми обрали фреймворк PyTorch – одну з провідних бібліотек глибокого навчання з відкритим кодом. PyTorch надає зручний Python-інтерфейс для створення нейронних мереж, автоматично обчислює градієнти і оптимізує параметри мережі з використанням стандартних методів. Важливо, що PyTorch дозволяє легко використовувати обчислення на GPU – досить перемістити модель і дані на відеокарту, після чого більшість операцій виконуватимуться там прискорено.

У нашій реалізації нейронна мережа визначена як клас SmallCRNN. Для спрощення організації навчального циклу ми задіяли надбудову PyTorch Lightning [16]. Це високорівневий фреймворк, що дозволяє винести стандартну логіку тренування (цикли епох, обробка батчів, обчислення метрик, збереження чекпоїнтів) у готові компоненти, зосередившись на власне визначенні моделі та даних.

Lightning дозволив також легко налаштувати колбеки: ми використали Early Stopping для зупинки навчання при стагнації метрики та Model Checkpoint для збереження найкращої моделі за найнижчим значенням помилки на валідації. Фреймворк автоматично зберіг найкращий чекпоїнт мережі.

Для оцінки якості розпізнавання символів ми скористалися метрикою Character Error Rate – це відношення кількості помилок в розпізнаних символах до загальної кількості символів у еталонному тексті. В PyTorch

Lightning є готовий модуль CharErrorRate у складі torchmetrics.text. У процесі валідації ми виконували greedy-декодування – тобто брали для кожної позиції символ з максимальною ймовірністю на виході мережі та формували з них рядок, прибираючи повтори та спеціальний символ blank. Цей рядок порівнювався з правильним рядком, і обчислювалась кількість вставок/видалень/заміни символів, необхідних для перетворення одного в інший. Накопичена по всіх прикладах помилка CER логувалася та використовувалася як ключова метрика для контролю навчання.

Зв’язка PyTorch та Lightning значно спростила експерименти з моделлю: ми могли змінювати архітектуру, не турбуючись про обгортку тренувального циклу, автоматично отримували логи, збереження моделі та навіть вбудовані підказки. Lightning виводив структуру моделі з кількістю параметрів, попереджав про потенційні проблеми з продуктивністю, наприклад недостатню кількість num_workers у DataLoader. Завдяки цьому процес навчання пройшов гладко.

Зазначимо також, що для формування пакетів даних ми використали модулі torch.utils.data.Dataset і DataLoader. Було створено власний клас IAMWordDataset , який завантажує з диску зображення слова та відповідний текст. Усередині нього застосовано бібліотеку Albumentations для аугментації: випадкові масштабування, повороти, еластичні викривлення, додавання шуму тощо – це допомагає моделі побачити кожен приклад у різних варіаціях і не переобладнатися [17]. При завантаженні зображення воно приводилось до стандартного розміру висоти з збереженням пропорцій ширини, потім інвертувалося і перетворювалося на зручне слово для розпізнавання. Таким чином, датасет на виході дає пару: тензор зображення та тензор послідовності кодів символів. DataLoader упаковував ці пари в батчі по 32 штуки і здійснював їх підготовку для GPU.

Використання інструментів PyTorch/Lightning забезпечило зручність і надійність при реалізації алгоритмів глибокого навчання, дозволивши зосередитись саме на архітектурі моделі та аналізі результатів.

2.5 Веб-фреймворк Flask для створення серверної частини

Для забезпечення доступності розробленої моделі розпізнавання рукописного тексту через веб-інтерфейс, було прийнято рішення реалізувати серверну частину у вигляді API. Вибір зупинився на мікрофреймворку Flask, який є легким, гнучким та популярним інструментом для розробки веб-додатків та API на мові Python.

Привабливість Flask для даного проекту обумовлена кількома факторами. Його простота та мінімалізм дозволяють швидко розгорнути функціональний веб-сервіс без надлишкової складності. Flask надає необхідний базовий інструментарій для маршрутизації HTTP-запитів, обробки даних та формування відповідей, не нав'язуючи жорстку структуру, що спрощує інтеграцію з уже існуючим кодом моделі машинного навчання. Це дозволяє уникнути складнощів, пов'язаних із взаємодією компонентів, написаних на різних мовах програмування, на серверній стороні. Крім того, Flask характеризується значною розширюваністю завдяки великій кількості доступних доповнень, які за потреби можуть додати функціонал для роботи з базами даних, автентифікації користувачів, налаштування CORS та інших аспектів.

У контексті даного проекту, Flask відіграє центральну роль у створенні кінцевих точок (endpoints) API, які слугують мостом між користувацьким інтерфейсом та моделлю розпізнавання. Основною є кінцева точка, умовно названа `/upload`. Цей маршрут налаштований на прийом HTTP POST-запитів, що містять зображення рукописного тексту. Зображення може передаватися у вигляді файлу або, наприклад, як рядок у кодуванні base64. Після отримання такого запиту, серверний додаток на Flask відповідає за попередню обробку вхідних даних та передачу підготовленого зображення на вхід навченій моделі PyTorch для виконання процедури розпізнавання. Отриманий від моделі текстовий результат потім форматується, зазвичай у структурований вигляд JSON, та повертається клієнту у тілі HTTP-відповіді.

2.6 Клієнтський інтерфейс на основі бібліотеки React

Для забезпечення зручної та інтерактивної взаємодії користувача із системою розпізнавання рукописного тексту було розроблено клієнтський веб-інтерфейс з використанням бібліотеки React. Вона розроблена компанією Facebook та є однією з провідних JavaScript-бібліотек для побудови динамічних користувацьких інтерфейсів і широко застосовується для створення односторінкових додатків.

Вибір React для фронтенд-частини проекту ґрунтується на його ключових перевагах. Однією з головних є компонентний підхід до розробки інтерфейсів. Ця парадигма дозволяє розбивати складний UI на менші, незалежні та повторно використовувані компоненти. У контексті системи розпізнавання тексту, це означає можливість створення окремих компонентів для таких функцій, як завантаження файлу зображення, попередній перегляд обраного зображення, відображення області для результатів розпізнавання, а також кнопок для ініціювання процесу розпізнавання та очищення форми. Такий підхід не тільки спрощує розробку та тестування, але й полегшує подальшу підтримку та модифікацію інтерфейсу.

Іншою важливою особливістю React є його декларативний стиль програмування. Розробник описує, як інтерфейс має виглядати в залежності від поточного стану додатку, а React бере на себе відповідальність за ефективне оновлення та рендеринг лише тих частин DOM, які зазнали змін. Це значно спрощує управління станом UI та зменшує кількість помилок. Використання віртуального DOM є ще однією технологічною перевагою React, що сприяє високій продуктивності. React спочатку вносить зміни у легковагову копію DOM в пам'яті (віртуальний DOM), а потім ефективно синхронізує ці зміни з реальним DOM браузера, мінімізуючи прямі та дорогі маніпуляції. Крім того, React має розвинену екосистему та активну спільноту, що забезпечує доступ до великої кількості готових бібліотек та інструментів для вирішення різноманітних завдань, таких як управління

станом (наприклад, Redux, Zustand або вбудований Context API), маршрутизація на стороні клієнта та виконання HTTP-запитів до сервера (рисунок 2.3).

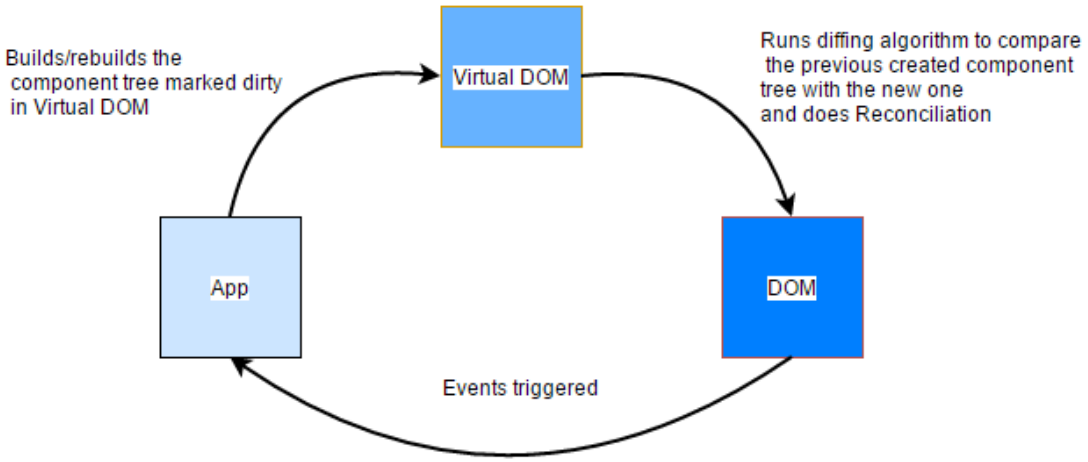

Рисунок 2.3 – Приклад роботи Virtual DOM

У даному проекті, React-додаток виконує декілька ключових функцій. Він надає користувачеві інтуїтивно зрозумілий спосіб завантаження зображення, що містить рукописний текст, з локального пристрою, а також забезпечує візуалізацію завантаженого зображення для перевірки. Після цього, React-компонент відповідає за підготовку та відправку цього зображення на серверний API, розгорнутий на Flask, для обробки моделлю розпізнавання. Після отримання відповіді від сервера, що містить розпізнаний текст, фронтенд-додаток динамічно відображає цей результат користувачеві у призначеній для цього області інтерфейсу. Ця взаємодія забезпечує плавний та швидкий користувацький досвід, дозволяючи легко отримувати результати роботи системи розпізнавання.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Передобробка даних та сегментація тексту

На початковому етапі практичної реалізації була виконана детальна передобробка зображень рукописного тексту, щоб підготувати їх до навчання моделі. Як вже зазначалося, для експериментів використовувалися сторінки з датасету IAM. Щоб впевнитись у правильності вибору методів передобробки, ми здійснили їх апробацію на одній зі сторінок.

3.1.1 Бінаризація зображення

Початкове зображення сторінки містить відтінки сірого – текст написаний темним кольором на світлому тлі, але фон не ідеально білий і може мати сірий тон або артефакти. Тому першим кроком ми перетворили зображення на чорно-біле. Для цього застосовано алгоритм Оцу, що автоматично обирає оптимальний поріг яскравості. Всі пікселі темніші за поріг стали чорними, а світліші – білими. Додатково виконано інвертування: тепер текст зображений білими пікселями, а фон – чорними. Нагадаємо, це зроблено для зручності подальшої роботи нейромережі – так їй легше сприймати активні області тексту як одиниці. На виході одержано бінарне зображення, де рукописні рядки чітко виділяються на контрастному фоні.

Звичайно, перед бінаризацією ми провели згладжування дрібних шумів гаусівським фільтром. Так було прибрано випадкові піксельні артефакти, які могли б заважати при пороговій обробці. Наприклад, окремі темні точки на фоні могли бути помилково розпізнані як частини символів. Після фільтрації текст залишився майже незмінним, але дрібні шуми зникли.

Снизу показано результат бінаризації: фон став повністю чорним, а текст – білим. Видно, що всі рядки читабельні, символи не злиті з фоном. Це

підтверджує, що обраний поріг спрацював оптимально для даного зображення (рисунок 3.1).

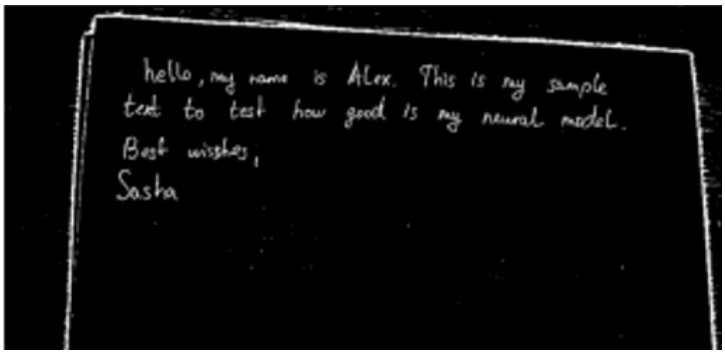

Рисунок 3.1 – Бінаризоване зображення

3.1.2 Сегментація на рядки

Використовуючи бінаризоване зображення, ми обчислили горизонтальну проєкцію – графік кількості білих пікселів по кожному горизонтальному рядку матриці зображення. На ділянках, де є текст, проєкція давала високі значення, а там, де між рядками – значення було близьким до нуля. Простежуючи цю проєкцію, алгоритм виявив декілька інтервалів по осі Y, в яких проєкція перевищує певний поріг. Кожен такий інтервал інтерпретовано як межі рядка тексту.

Візуальна перевірка підтвердила коректність роботи алгоритму: всі рядки тексту були охоплені, «пустих» виділень нема, і навпаки – не залишилось ні одного текстового рядка, який алгоритм пропустив би.

3.1.3 Сегментація рядків на слова

Далі обробка проводилась для кожного вирізаного фрагменту рядка окремо. На кожному рядку ми розрахували вертикальну проєкцію – кількість білих пікселів в кожній колонці цього фрагменту. На ділянках, де є слово,

проєкція буде значною, а між словами – стовпчики проєкції майже нульові. За таким профілем ми розрізали рядок на окремі сегменти.

Алгоритм аналізував послідовність проєкції: якщо значення стовпця падає до нуля і залишається низьким протягом певної ширини, то між двома сусідніми словами достатній проміжок – фіксуємо кінець слова. Наступний стовпець, де проєкція знову підніметься, стає початком нового слова. Важливо було відсікати надто дрібні виділення: приміром, один-два стовпці могли випадково мати невелике значення через якийсь виступ літери, але це ще не межа слова. Тому ми задавали мінімальну ширину сегмента, приймаючи лише ті проміжки, що ширші за кілька пікселів, як справжні роздільники слів. В результаті кожен рядок розбився на ряд зображень окремих слів.

3.1.4 Збереження та підготовка сегментів

Кожне виділене слово було збережено як окреме зображення у спеціально створеній теці. При збереженні ми інвертували значення назад, оскільки зручніше зберігати чорні символи на білому – такий формат звичніший і його легше проглядати, а для нейронної мережі ми знову інвертуємо при завантаженні. Кожному файлу присвоєно унікальне ім’я, що включало номер сторінки та координати сегмента. Паралельно формувався CSV файл, де для кожного такого зображення вказано його шлях та текст.

Наприкінці передобробки ми отримали структуровані дані: папку з тисячами зображень слів і таблицю, що зіставляє кожному зображенню його правильний текст. Ці дані стали основою для навчання нейронної мережі.

Варто підкреслити, що сегментація – критичний етап, від якого залежить якість навчального матеріалу. Якщо б деякі слова були неправильно нарізані, наприклад, частина сусіднього слова потрапила б у сегмент, або слово порізалось на половини, то модель отримала б некоректні приклади для навчання, що могло б серйозно знизити точність. У нашому випадку,

завдяки простоті даних IAM (чіткі горизонтальні рядки, достатні проміжки між словами) і застосованим перевіркам, сегментація пройшла успішно: за оцінками, понад 99% сегментів відповідали реальним словам без зміщення меж.

Отримані приклади зображень слів різноманітні за довжиною – від дуже коротких, наприклад, слово «І» з однієї літери до довгих слів. Це не проблема для моделі, оскільки ми будемо застосовувати підхід з адаптивною довжиною, який дозволяє обробляти послідовності змінної довжини.

Підсумовуючи, етап передобробки забезпечив чисті та зручні для аналізу дані: кожне зображення слова однорідне за яскравістю, очищене від зайвого фону, а весь набір нарізано на логічні слова з відомими мітками. Це створило міцний фундамент для наступного етапу – навчання нейронної мережі розпізнаванню цих слів.

3.2 Архітектура нейронної мережі для розпізнавання тексту

Як було розглянуто в теоретичній частині, ефективним підходом для такої задачі є поєднання згорткових та рекурентних шарів нейронної мережі з використанням CTC-алгоритму. На основі цієї ідеї було спроєктовано компактну модель типу CRNN – умовно назвемо її SmallCRNN. Вона забезпечує баланс між якістю та швидкістю інференсу на обмежених обчислювальних ресурсах. Складається модель з двох основних блоків:

- Згортковий блок (CNN) – для екстракції ознак із зображення; у SmallCRNN це каскад із п’яти згорток 3 x 3 з наростаючою кількістю фільтрів, що стискає кадр до компактного тензора ознак 8 x 32 x 256.
- Рекурентний блок (RNN) – для обробки послідовності цих ознак та генерації вихідної послідовності символів; реалізований як двошарова двонапрямна LSTM на 256 станів, яка враховує контекст з обох боків перед подачею на CTC (рисунок 3.2).

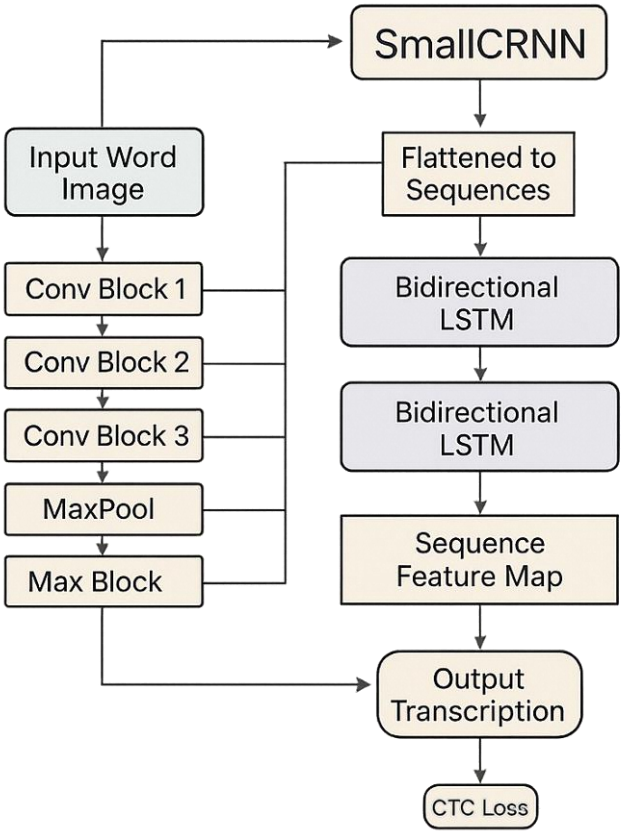

Рисунок 3.2 – Архітектура нейронної мережі SmallCRNN

3.2.1 Підготовка зображення

Перш ніж слово пройде через мережу, його зображення спрощується: перетворюється на чорно-біле, фон стає чорним, символи – білими. Висоту кадру приводять до 32 пікселів, тоді як ширина залишається довільною, бо залежить від довжини конкретного слова. У результаті утворюється охайна вузька «стрічка», готова до подальшої обробки.

3.2.2 Згортковий блок – цифровий сканер

Далі зображення потрапляє до каскаду згорткових шарів, які поводяться як електронний сканер. Кожен шар поступово зменшує висоту «стрічки» (від 32 пікселів до одного), водночас збільшуючи кількість цифрових ознак, що описують форму штрихів, заокруглення та інші деталі

літер. Після кількох таких перетворень виходить довга послідовність числових векторів: кожен відповідає маленькому вертикальному зрізу слова й містить інформацію про те, що саме «бачить» мережа в цьому фрагменті.

3.2.3 Рекурентний блок – внутрішній читач

Утворену послідовність векторів бере на себе двонапрямлена LSTM-мережа. Вона читає рядок одночасно зліва направо й справа наліво, тому враховує контекст обох боків. Це допомагає точніше розпізнати неоднозначні ділянки, наприклад відрізнити «m» від «n». Після LSTM для кожної позиції формується набір ймовірностей усіх можливих символів. Алгоритм CTC узгоджує цю довгу послідовність із реальним написанням слова: він автоматично вирівнює символи, додає потрібні пропуски й навчає мережу правильно «розтягувати» або «стискати» літери вздовж зображення без ручної розмітки їх меж.

3.2.4 Підсумок роботи мережі

Уся система містить приблизно дев’ять мільйонів параметрів — це дещо, що сучасна відеокарта здатна швидко опрацювати. Після навчання одного слова достатньо кількох десятків мілісекунд, щоб пройти шлях від вихідної картинки до готового тексту. Так без жодного ручного нарізання символів модель самостійно читає рукописні слова й видає їх у зручному цифровому вигляді.

3.3 Навчання моделі та результати розпізнавання

Навчання проводилося протягом 250 епох з використанням алгоритму AdamW (варіація стохастичного градієнтного спуску) при початковому темпі навчання 1e-3. Було налаштовано ранню зупинку: якщо валідаційна помилка

не покращувалася протягом 3 епох, навчання припинялося достроково, щоб запобігти перенавчанню. Також застосовувався планувальник швидкості навчання.

Під час тренування ми спостерігали стабільне зниження значення функції втрат CTC на тренувальному та валідаційному множинах. Вже після перших 2 епох мережа навчилася розпізнавати найпростіші символи та короткі слова (значення CER впало з ~100% до ~30%). Далі темпи покращення сповільнились, але ще кілька десятків епох дозволили покращити результат. Навчання було зупинено достроково на 224-й епосі – це сталося після того, як валідаційний CER не зменшувався 2-3 епохи поспіль (рисунок 3.3). Найкраща зафіксована метрика на валідації склала CER = 0.226. Це означає, що приблизно 22-23 символи зі 100 модель розпізнавала неправильно на валідаційному наборі.

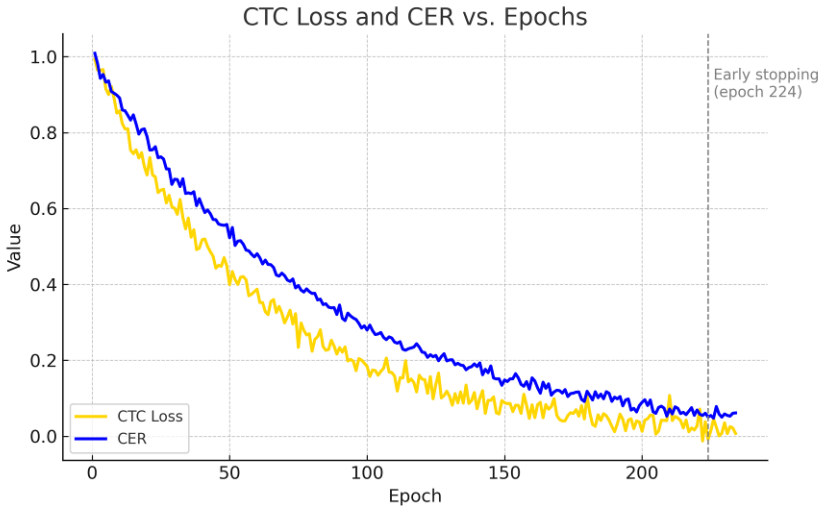

Рисунок 3.3 – Графіки CTC-loss і CER по епохах

На перший погляд, помилка 22% видається значною, але слід врахувати складність даних: модель вгадувала кожен символ незалежно, спираючись лише на свій тренінг. Деякі символи, а особливо великі літери чи рідковживані знаки траплялися нечасто, тому їх розпізнавання могло давати похибку. Також певні почерки були важчими для моделі.

3.3.1 Аналіз і приклади результатів

Для якісної оцінки ми протестували фінальну модель на декількох прикладах з тестового набору, який модель не бачила зовсім під час тренування. В якості тесту брали цілісні рядки тексту: подавали на вхід скановану рукописну строку, сегментували її на слова та кожне слово пропускали через модель, а потім з'єднували результати. Нижче наведено кілька прикладів фактичної роботи системи:

− Приклад 1: Рядок з оригінальним текстом «sure, he, during, of, booty». Модель видала результат: «sure, he, during, of, booty». Видно, що всі слова розпізнані правильно.

− Приклад 2: Оригінал: «January 7, 1885, in New York City.». Розпізнано як «January 7, 1885, in New York City .» – тобто точно, символ в символ. Модель успішно впоралася з цифрами, комами, великою літерою «J» та іншими складовими. Це демонструє, що на знайомих паттернах мережа помиляється рідко.

− Приклад 3: Оригінал: «He quickly realized that the scheme was interesting». Модельний результат: «He quiclky realized that the schene was interesting». Тут два помітних недоліки: слово «quickly» розпізнано як «quiclky» (модель переплутала порядок літер c і k, фактично зробивши перестановку – можливо, через почерк, де ці літери наклалися) та слово «scheme» прочитане як «schene» (літеру m розпізнано як n). Такі помилки характерні: «m» і «n» мають схожий патерн, відрізняючись лише додатковим штрихом, який модель могла інтерпретувати інакше. Перестановка «ck» – ці літери в рукописі могли злитися і модель не точно визначила межу між ними. Загалом, помилки є, але вони не катастрофічні: сенс речення ще можна зрозуміти. CER для цього прикладу: 2 помилки на ~30 символів 6.7%.

− Приклад 4 (складніший почерк): Оригінал: «committee on appropriations, and urged». Результат: «cornmittee on appropriations, and urged». Модель невпевнено впізнала першу букву слова «committee» – почерк був

такий, що «co» вона інтерпретувала як «corn». Решта слова правильно. Ця помилка може бути пояснена тим, що почерк автора міг бути менш типовим або ж слово «committee» – довге і рідкісне, тому мережі складно. Тим не менш, решта тексту зчитана майже без зауважень (рисунок 3.4).

Рисунок 3.4 – Приклад успішного розпізнавання

В цілому модель краще розпізнає короткі часті слова (службові: «and», «the», «in», займенники, прийменники) – їх вона бачила тисячі разів під час навчання. Довгі ж і незвичні слова іноді містять 1-2 помилки. Найскладніше моделі далися власні імена, скорочення або слова з рідкісними літерами. Так, спостерігалось, що великі літери «G», «Y», «Z» вона могла плутати між собою через схожість форм. Була створена теплова карта помилок (рисунок 3.5)

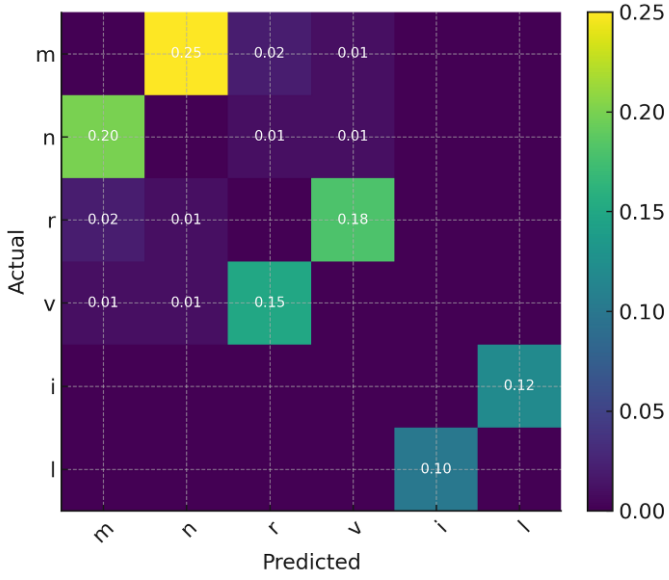

Рисунок 3.5 – Теплова карта плутаних символів

3.3.2 Оцінка якості на рівні слов та символів

На валідаційному наборі CER 22.6%. Якщо перерахувати це на метрику WER (Word Error Rate – відсоток слів з хоча б однією помилкою), то виходить значення близько 35-40%. Це означає, що приблизно третина слів містили хоча б одну помилку в розпізнаванні. Знову ж, більшість цих помилок – це один неправильний символ у слові. Тобто часто слово помилкове мінімально (напр. «brothe» замість «brother»).

Для порівняння, сучасні передові моделі (більші за розміром і навчені з мовними моделями) мають WER ~10-15% [18]. Наш результат 22.6% трохи гірший, але цілком очікуваний для компактної моделі, яка виконується на не дуже потужній локальній машині.

3.3.3 Збереження та використання моделі

Найкращий стан мережі було збережено у файл чекпоїнта. Для зручності подальшого використання ми завантажили цю модель та встановили її в режим оцінювання model.eval(). Тепер будь-яке нове зображення слова можна подати на вхід, викликати model(img_tensor) і отримати вихідні логіти. Розроблено функцію greedy_decode , яка перетворює логіти на текст, реалізуючи описаний вище простий підхід (вибирає argmax по кожній позиції та фільтрує повтори і blank). Ця функція видала рядок символів – результат розпізнавання.

У підсумку, результати розпізнавання можна вважати вдалими з точки зору демонстрації принципу роботи системи. Модель навчилася перетворювати зображення рукописних слів у текст з прийнятною якістю. Виявлені помилки вказують напрями, за якими систему можна покращувати, але навіть у поточному стані вона здатна автоматично прочитати значну частину тексту правильно. Це підтверджує доцільність застосованих технологій та методик для поставленої задачі.

4 ІНСТРУКЦІЯ КОРИСТУВАЧА

4.1 Загальний огляд інтерфейсу

Веб-інтерфейс системи розроблено з метою забезпечення простої та інтуїтивно зрозумілої взаємодії. Головна сторінка умовно поділена на кілька функціональних блоків: центральна область для завантаження файлів, панель з налаштуваннями розпізнавання, динамічна область для відображення результатів, а також панель управління користувачем у правому верхньому куті для реєстрації та входу (рисунок 4.1).

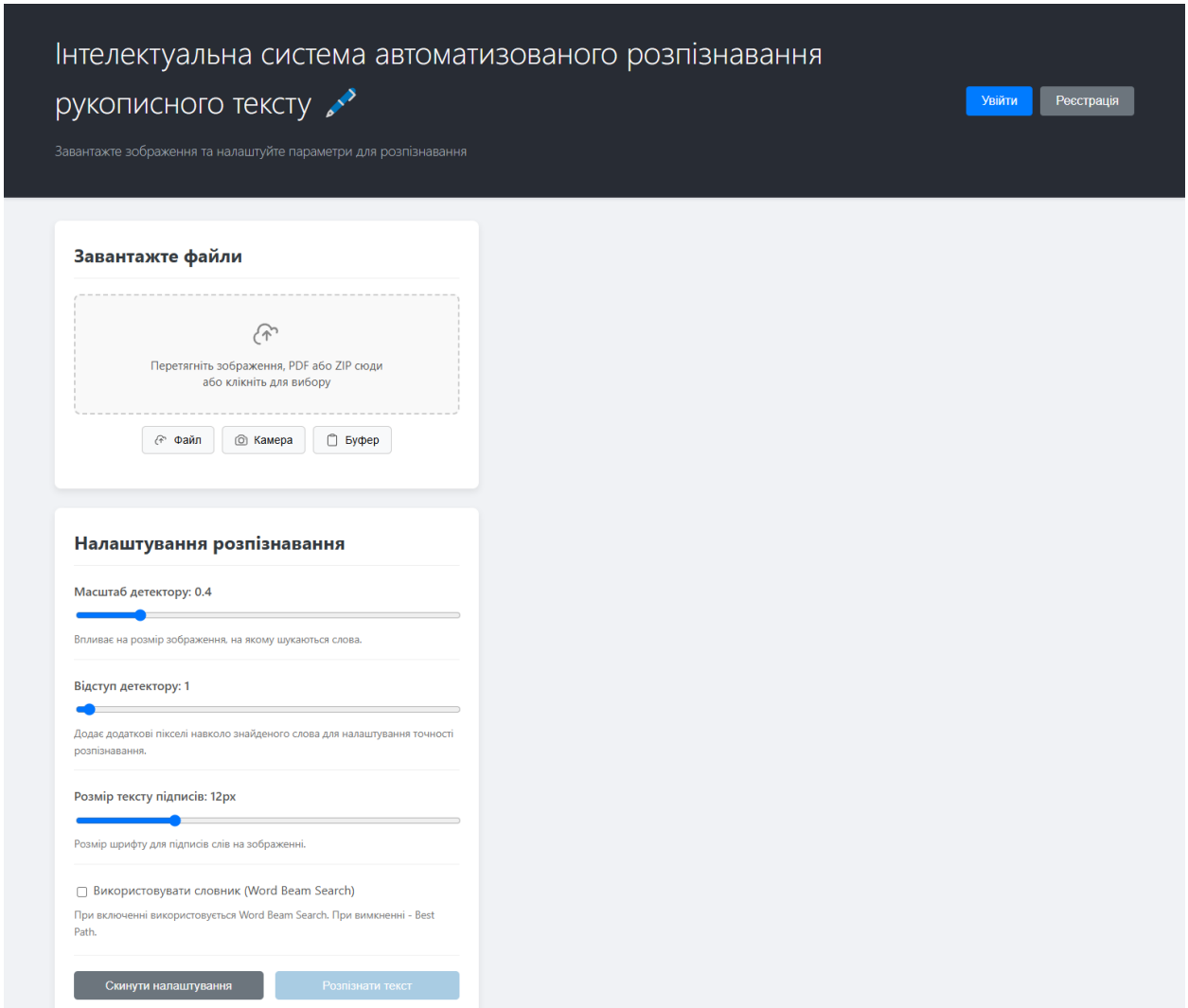

Рисунок 4.1 – Інтерфейс системи

4.2 Завантаження вихідного файлу

Робота із системою починається із завантаження документа. Для цього передбачено декілька зручних методів:

− Завантаження файлу: Користувач може натиснути на кнопку «Файл» або просто перетягнути документ у спеціальну область. Система підтримує різні формати, включаючи зображення, PDF-документи та ZIP-архіви з кількома файлами.

− Використання камери: Опція «Камера» активує веб-камеру пристрою, дозволяючи зробити миттєвий знімок рукописного тексту, наприклад, нотаток у зошиті.

− Вставка з буфера обміну: Кнопка «Буфер» дозволяє вставити зображення, яке було попередньо скопійоване в буфер обміну (наприклад, знімок екрана).

Для прикладу завантажимо JPG-файл сторінки, яка містить рукописний текст. Можемо побачити попередній перегляд файлу (рисунок 4.2).

Рисунок 4.2 – Завантажене фото з рукописним текстом

4.3 Конфігурація параметрів розпізнавання

Для досягнення максимальної точності користувач може гнучко налаштувати процес розпізнавання за допомогою кількох параметрів. Ці налаштування безпосередньо впливають на етапи сегментації тексту та його подальшої інтерпретації моделлю.

− Масштаб детектора: Цей параметр контролює роздільну здатність, з якою модель аналізує зображення. Для документів з дуже дрібним або щільним текстом може бути корисним зменшити масштаб, щоб алгоритм міг краще розрізняти окремі елементи, і навпаки.

− Відступ детектора: Параметр додає запас у пікселях навколо кожної виявленої межі слова. Це критично важливо для рукописів з курсивним або похилим письмом, де частини літер можуть виходити за формальні межі слова. Достатній відступ гарантує, що слово буде захоплене повністю.

− Розмір тексту підписів: Це виключно візуальне налаштування, яке змінює розмір текстових міток, що відображаються над розпізнаними словами у вікні попереднього перегляду (рисунок 4.3).

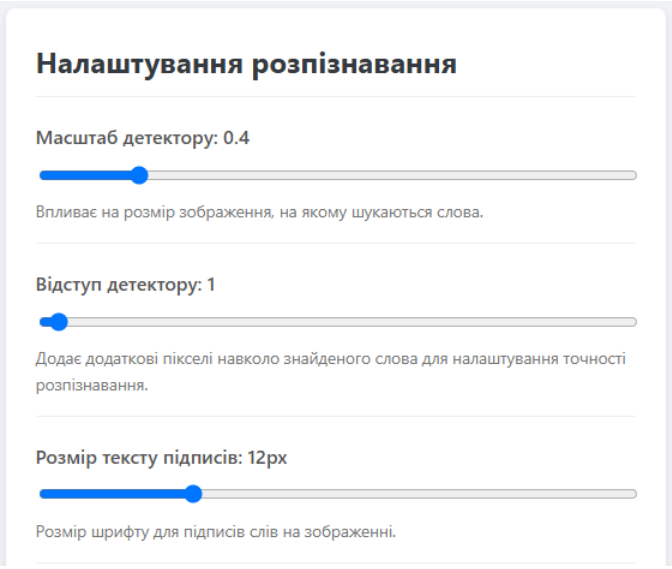

Рисунок 4.3 – Основні налаштування розпізнавання тексту

Також користувач має доступ до дуже важливої функції – використовувати словник (Word Beam Search). Це ключова функція для значного підвищення точності. Її активація вмикає двоступеневий процес. Спочатку нейронна мережа CRNN виконує базове розпізнавання на рівні окремих символів. Потім алгоритм Word Beam Search аналізує отриману послідовність символів і, спираючись на вбудований словник, коригує її, обираючи найімовірніші реальні слова. Це дозволяє ефективно виправляти поширені помилки, такі як плутанина між візуально схожими літерами (рисунок 4.4).

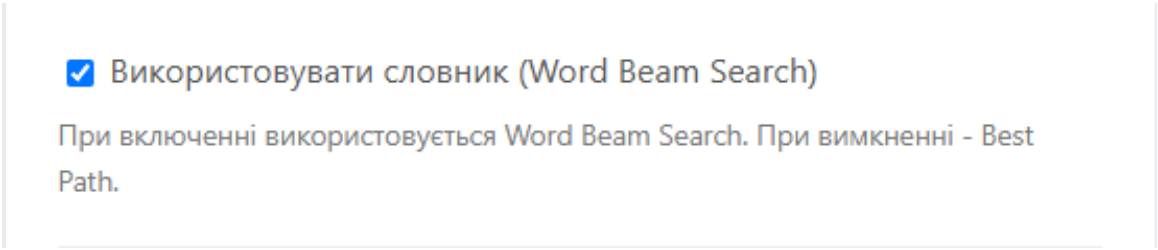

Рисунок 4.4 – Word Beam Search

У нижній частині сервісу залишаються дві кнопки. Перша дозволяє скинути налаштування до стандартних. Друга основна кнопка розпізнає текст. На неї й натискаємо (рисунок 4.5).

Рисунок 4.5 – Основні кнопки

4.4 Аналіз результатів

Після натискання кнопки «Розпізнати текст» система обробляє дані та представляє результат у двох форматах (рисунок 4.6).

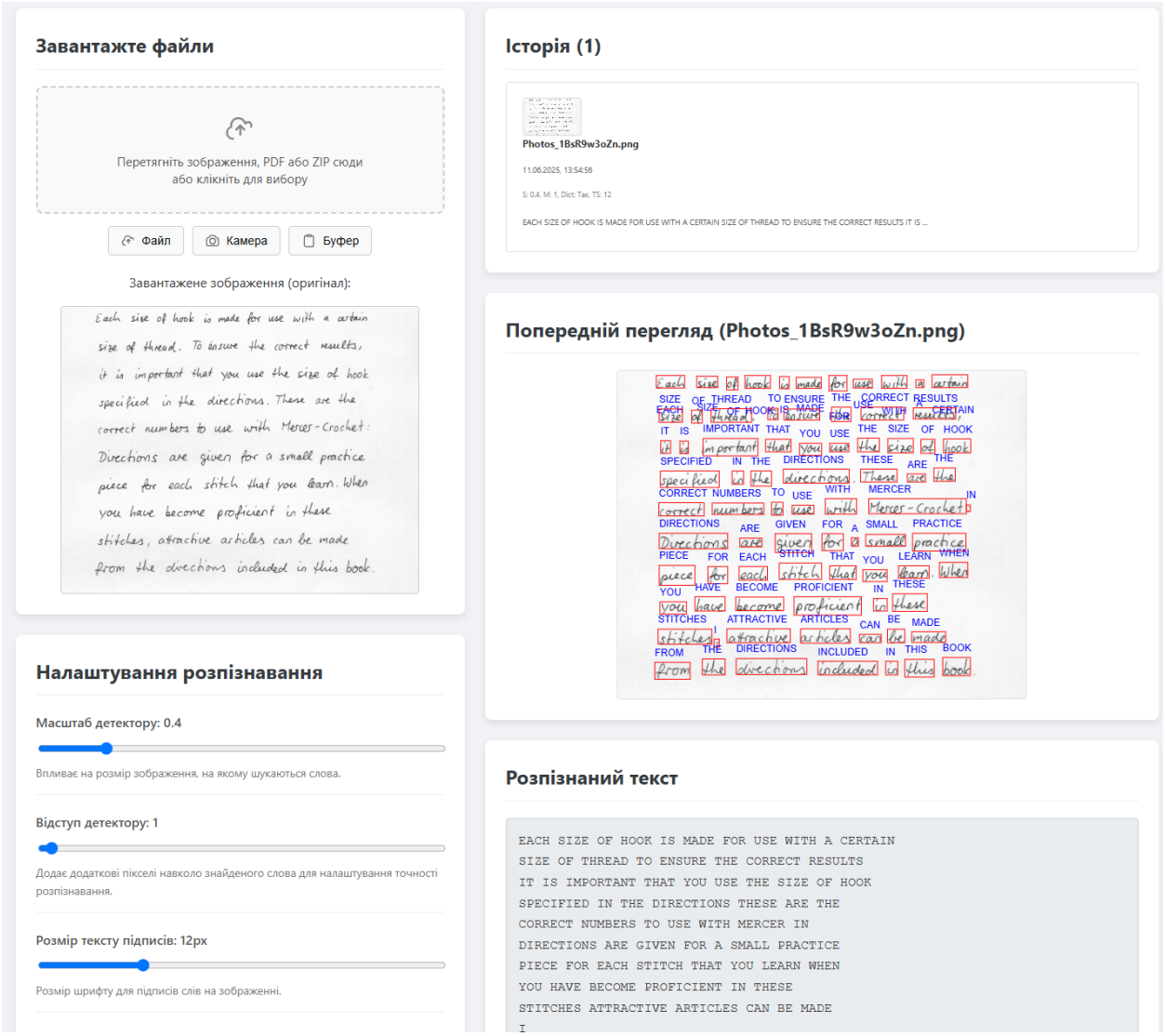

Рисунок 4.6 – Результат розпізнаного тексту

У вікні попереднього перегляду відображається вихідне зображення, на яке накладено червоні рамки. Ці рамки візуалізують роботу алгоритму сегментації і точно показують, які частини зображення були ідентифіковані як окремі слова перед подачею на розпізнавання. Це дозволяє користувачеві

оцінити, наскільки коректно система розділила текст, і за потреби змінити налаштування для кращого результату. Нижче розташоване текстове поле, що містить фінальну версію тексту після обробки, а також кнопки для завантаження у форматі .txt або .doc (рисунок 4.7)

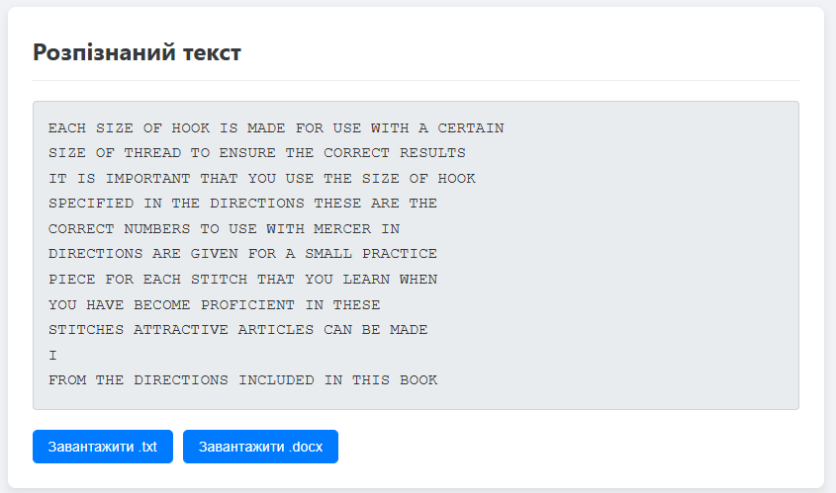

Рисунок 4.7 – Блок з розпізнаним текстом

Також, якщо користувач не зареєстрований, попередньо розпізнані зображення тимчасово зберігаються у блоці «Історія» зверху. Користувач може натискати на елементи в цьому блоці та переключатися між розпізнаними файлами, які, у свою чергу, зберігають налаштування перед розпізнаванням тексту (рисунок 4.8).

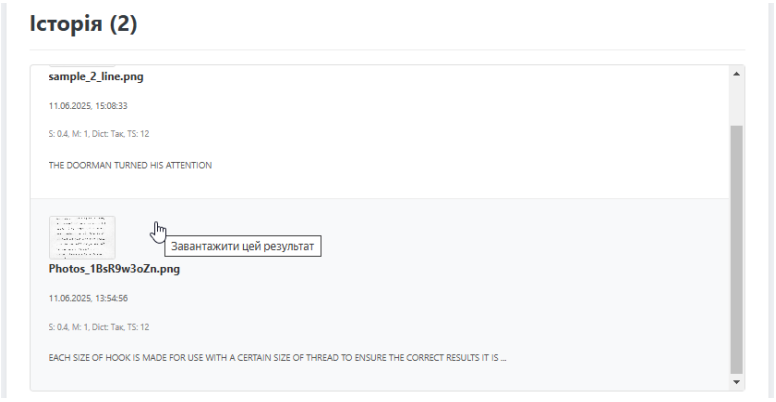

Рисунок 4.8 – Блок «Історія»

Як відомо, PDF-файли мають відразу кілька зображень у собі. Для цього у проекті було реалізовано також попередній перегляд цих зображень. Наприклад, завантажимо PDF-документ, якій містить зображення з рукописним текстом і натиснемо «Розпізнати текст» (рисунок 4.9).

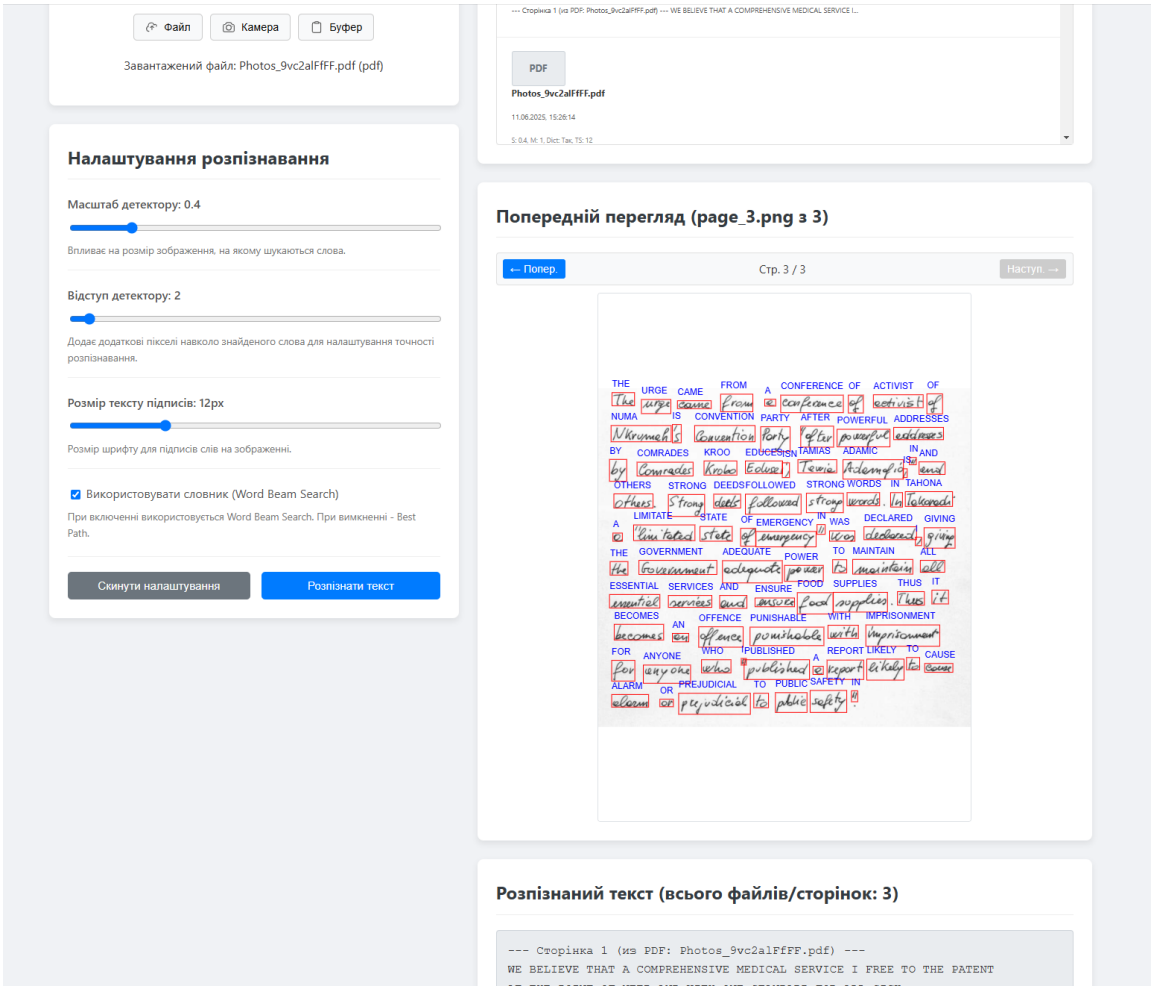

Рисунок 4.9 – Попередній перегляд PDF-документу

Можемо бачити, що сервіс дає можливість переглядати сторінки документу, дозволяючи зручно бачити як точно сервіс розпізнає текст та змінити налаштування, якщо це доречно. Так само система працює і з zip-архівами, які містять в собі зображення, допускаючи можливість попереднього перегляду цих зображень. Розпізнаний текст автоматично поділяється на сторінки відповідно до вихідного документа (рисунок 4.10).

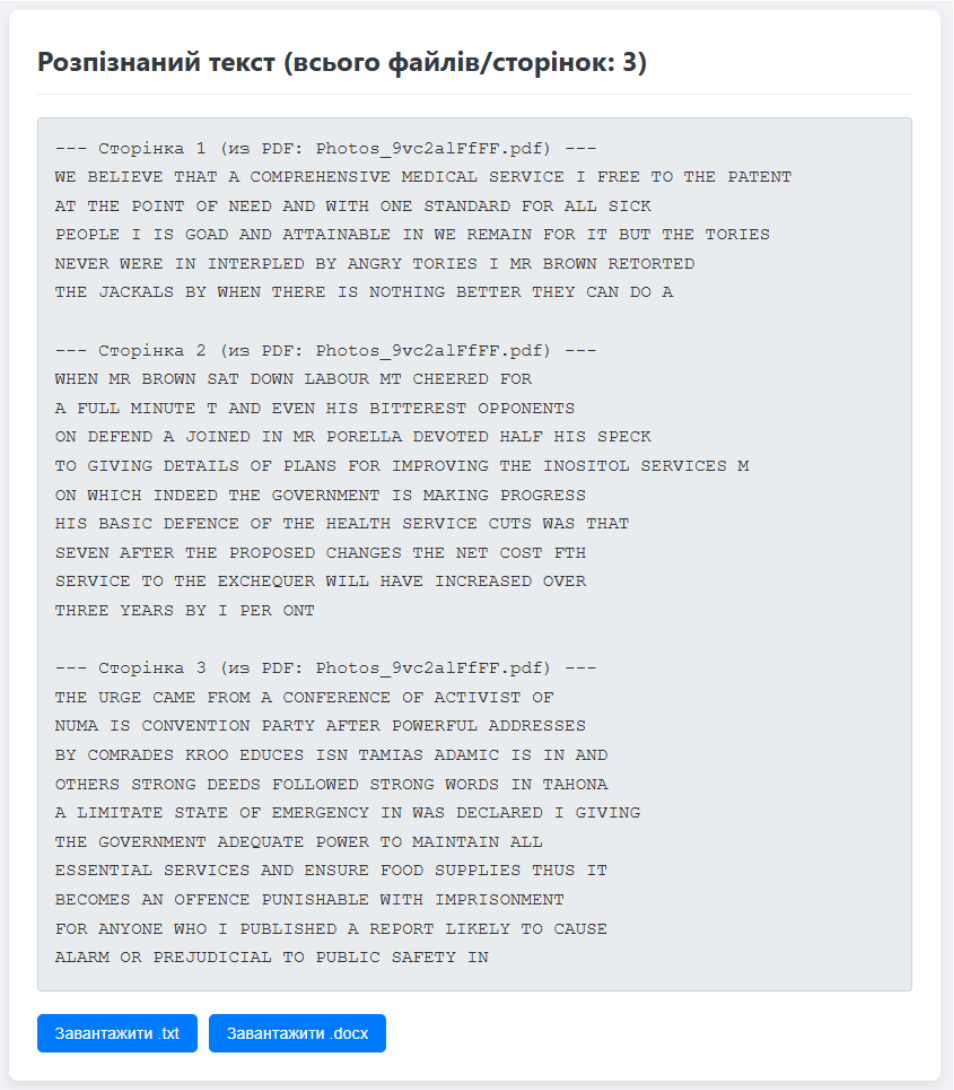

Рисунок 4.10 – Попередній перегляд PDF-документу

Якщо буде завантажено пошкоджений файл, або файл з несумісним розширенням, або у ZIP-архіві будуть файли, які міститимуть некоректні файли, або відсутня камера для з’йомки, сервіс видатиме відповідну помилку (рисунок 4.11).

Рисунок 4.11 – Приклад помилки при завантаженні невірного типу файлу

4.5 Управління обліковим записом

Система пропонує додатковий функціонал для зареєстрованих користувачів. За бажанням, користувач може створити свій обліковий запис. Для цього потрібно у правому верхньому кутку натиснути на кнопку «Реєстрація» або «Увійти». Відкриється форма для створення нового акаунту (рисунок 4.12).

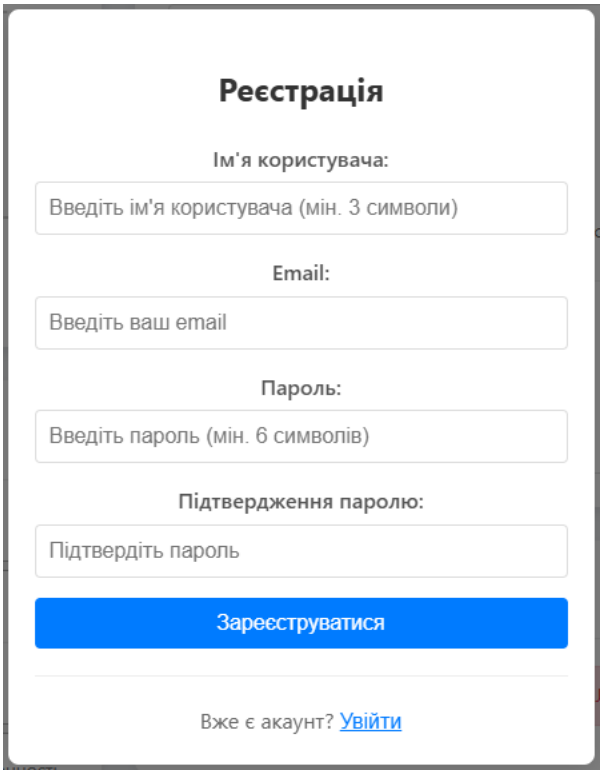

Рисунок 4.12 – Форма реєстрації акаунту

Важливо зазначити, що форма не просто приймає будь-який текст, а виконує валідацію даних у реальному часі та після спроби відправки. Цей процес включає перевірку на мінімальну довжину логіну, довжину та складність паролю (мінімально 1 цифра та велика літера), а також правильний формат Email адреси. Крім того, виконується запит до бази даних, щоб переконатися, що таке ім'я користувача або пошта є унікальними і ще не використовуються іншим акаунтом (рисунок 4.13).

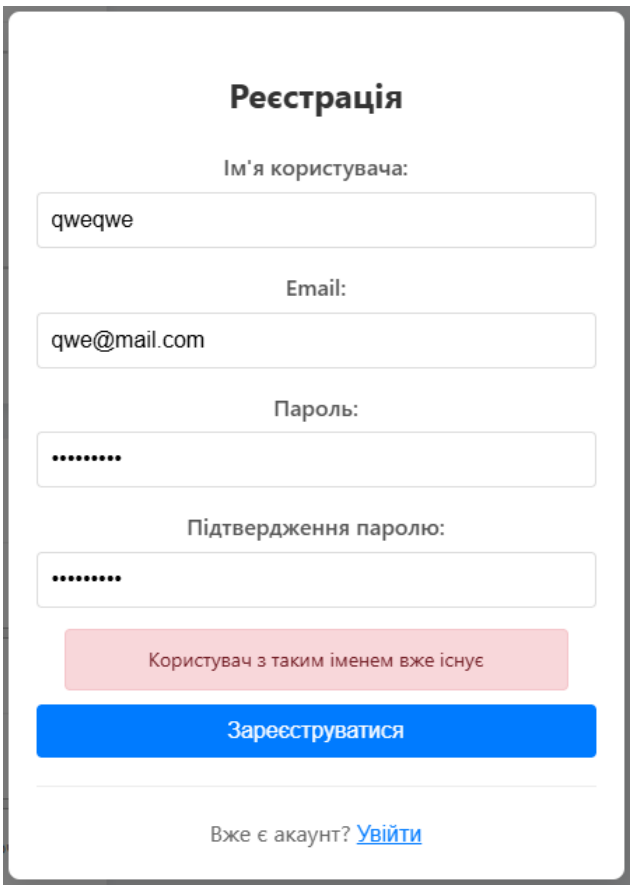

Рисунок 4.13 – Типова помилка

Після успішної реєстрації користувач отримує доступ до додаткового функціоналу в правому верхньому кутку інтерфейсу. Там розміщені два ключові елементи: кнопка виходу з акаунту і функціонал «Історія». Кнопка виходу меншого розміру і відрізняється за кольором, що допомагає уникнути випадкових натискань (рисунок 4.14).

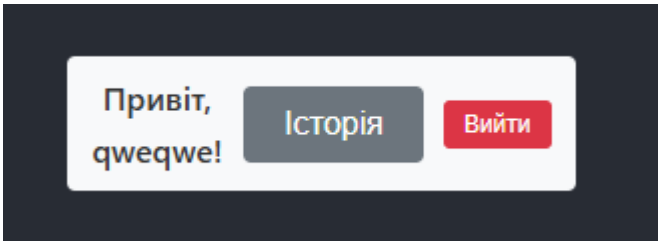

Рисунок 4.14 – Меню облікового запису

Основною функцією зареєстрованих користувачів є можливість зберігати історію в обліковому записі та відновлювати її у будь який момент. Всі розпізнані зображення та налаштування зберігаються на сервері та можуть бути отримані при вході в свій обліковий запис. У статистиці показано скільки всього файлів та сторінок було розпізнано та написано які типи файлів були завантажені. Також користувач може експортувати історію у форматі .csv або .json (рисунок 4.15).

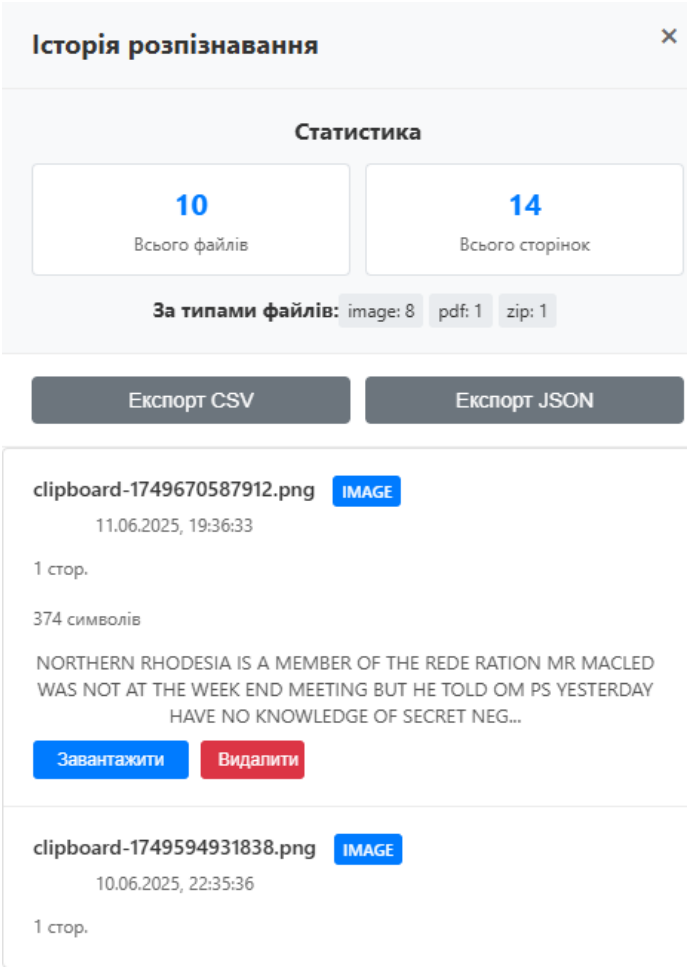

Рисунок 4.15 – Меню історії для зареєстрованих користувачів

ВИСНОВКИ

У ході виконання роботи було створено й досліджено прототип системи автоматичного розпізнавання рукописного тексту. Було вивчено еволюцію технологій OCR / HTR і встановлено, що традиційні шаблонні та статистичні методи, попри історичну значущість, мають суттєві обмеження при роботі з рукописами, тоді як сучасні згортково-рекурентні мережі демонструють набагато кращі результати завдяки здатності самостійно виокремлювати ознаки й ураховувати контекст.

На практичному етапі було підготовлено корпус IAM, виконано бінаризацію та проєкційну сегментацію, що забезпечило модель високоякісними вхідними зображеннями. Було спроєктовано компактну CRNN-архітектуру й застосовано алгоритм CTC, завдяки чому модель навчилась зіставляти послідовність ознак із послідовністю символів без ручної розмітки меж літер.

Було підтверджено важливість якісної передобробки зображень, оскільки саме вона мінімізує шум і спрощує задачу для мережі. Було виявлено, що відсутність мовного пост-процесингу призводить до накопичення дрібних помилок на рівні слів, тож інтеграція словника або мовної моделі розглядається як перспективний крок підвищення точності. Також було показано, що запропоноване рішення легко масштабувати: можливо збільшувати обсяг даних, адаптуватися під нові мови та оптимізувати модель для різних апаратних платформ.

Отримані результати засвідчили практичну цінність прототипу: уже нині його можна використати для чернеткового оцифрування нескладних рукописних документів – нотаток, формулярів, адрес. Загалом було доведено, що сучасні методи глибокого навчання відкривають реальні можливості для розв’язання складної проблеми читання рукописів і сприяють подальшій автоматизації обробки документів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Rabiner L. A tutorial on hidden Markov models and selected applications in speech recognition. Proceedings of the IEEE, 1989, 77(2): 257–286. URL: https://ieeexplore.ieee.org/document/18626 (дата звернення: 20.04.2025).

2. Li M., Lv T., Cui L. та ін. TrOCR: Transformer-Based Optical Character Recognition with Pre-Trained Models. Proc. AAAI, 2023, 37 (11): 13242-13250. URL: https://arxiv.org/abs/2109.10282 (дата звернення: 19.05.2025).

3. Marti U.-V., Bunke H. The IAM-database: an English sentence database for offline handwriting recognition, 2002, 39–46. (дата звернення: 22.04.2025).

4. Graves A., Fernández S., Gomez F., Schmidhuber J. Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. Proc. ICML, 2006, pp. 369–376. URL: https://www.cs.toronto.edu/~graves/icml_2006.pdf (дата звернення: 19.05.2025).

5. AlKendi W., Gechter F., Heyberger L., Guyeux C. Advancements and Challenges in Handwritten Text Recognition: A Comprehensive Survey. Journal of Imaging, 2024. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC10817575 (дата звернення: 05.05.2025).

6. OpenCV – Open-Source Computer Vision Library. URL: https://opencv.org (дата звернення: 18.05.2025).

7. Khan M., Nabi F., Ahmed S. Enhancing Optical Character Recognition on Images with Mixed Handwritten and Printed Text Using Semantic Segmentation. Sensors, 2021, 21 (21): 7306. DOI: 10.3390/s21217306. URL: https://www.mdpi.com/1424-8220/21/21/7306 (дата звернення: 19.05.2025).

8. Шатохін О., Ілюнін О. Інтелектуальна система для автоматизованого розпізнавання рукописного тексту, IX Міжнародна студентська наукова конференція, Україна. 2025. С. 108 – 109.

9. Kluyver T., Ragan-Kelley B., Pérez F. Jupyter Notebooks – a Publishing Format for Reproducible Computational Workflows. In Positioning and Power in Academic Publishing, IOS Press, 2016, pp. 87-90. (дата звернення: 19.05.2025).

10. Raina R., Madhavan A., Ng A. Large-Scale Deep Unsupervised Learning Using Graphics Processors. Proceedings of the 26th International Conference on Machine Learning (ICML), 2009, 873–880. URL: https://dl.acm.org/doi/10.1145/1553374.1553486 (дата звернення: 21.05.2025).

11. Paszke A., Gross S., Massa F. та ін. PyTorch: An Imperative Style, High-Performance Deep Learning Library. NeurIPS, 2019, pp. 8024-8035. URL: https://arxiv.org/abs/1912.01703 (дата звернення: 19.05.2025).

12. Garrido-Muñoz C., Rios-Vila A., Calvo-Zaragoza J. Handwritten Text Recognition: A Survey. arXiv, 2025. URL: https://arxiv.org/abs/2502.08417 (дата звернення: 28.04.2025).

13. Handwriting Text Recognition – Medium Blog (Analytics Vidhya). URL: https://medium.com/analytics-vidhya/handwriting-text-recognition-3712978249da (дата звернення: 17.05.2025).

14. Ureña J. M. Tesseract-OCR: Evaluating Handwritten Text Recognition. Medium, 2021. URL: https://joseurena.medium.com/tesseract-ocr-evaluating-handwritten-text-recognition-1c6db85b2e7f (дата звернення: 07.05.2025).

15. PyTorch Lightning Documentation. URL: https://lightning.ai/docs/pytorch (дата звернення: 20.05.2025).

16. Buslaev A., Parinov A., Khvedchenya E. та ін. Albumentations: Fast and Flexible Image Augmentations. Information, 2020, 11 (2): 125. DOI: 10.3390/info11020125. URL: https://www.mdpi.com/2078-2489/11/2/125 (дата звернення: 19.05.2025).

17. Li H., Zhang Y., Chen G. Handwritten Text Recognition with Vision Transformers. Pattern Recognition Letters, 2022, 165: 78-85. DOI: 10.1016/j.patrec.2022.12.001 (дата звернення: 19.05.2025).

18. Kundu S., Paul S., Sarkar R. Learning-Free Text Line Segmentation for Historical Handwritten Documents. Applied Sciences, 2020, 10 (22): 8276. URL: https://www.mdpi.com/2076-3417/10/22/8276 (дата звернення: 19.05.2025).