

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Мобільний додаток для розпізнавання рукописного
тексту»**

Виконав студент групи ПЗм-23-1
ЛЕУТ Назарій Віталійович

Керівник роботи: д.пед.н., професор
БАКАЛЮК Тетяна Анатоліївна

Рецензент:
ВЛАСЕНКО Олег Васильович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»
Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«01» жовтня 2024 р

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ЛЕУТ Назарій Віталійович**

Керівник роботи: **д.пед.н., професор ВАКАЛЮК Тетяна Анатоліївна**

Тема роботи: **«Мобільний додаток для розпізнавання рукописного тексту»**

Затверджена Наказом закладу вищої освіти від **«01» жовтня 2024 р., №520/с**

Вихідні дані для роботи: **дослідження сучасних методів обробки зображень і тексту для розпізнавання рукописного тексту, розробка мобільного додатку та серверної частини для автоматизації процесу оцифрування рукописних матеріалів; алгоритми попередньої обробки зображень, методи машинного навчання; інструменти розробки програмного забезпечення.**

Керівники / консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Керівник/консультант	Завдання видав	Завдання прийняв
1	Вакалюк Т.А.	25.01.2024	25.01.2024
2	Вакалюк Т.А.	01.11.2024	01.11.2024
3	Вакалюк Т.А.	01.12.2024	01.12.2024

РЕФЕРАТ

Кваліфікаційна робота магістра складається з дослідження методів розпізнавання рукописного тексту, розробки мобільного додатку для оцифрування рукописних матеріалів, створення серверної частини для обробки зображень і розпізнавання тексту, а також пояснювальної записки. Пояснювальна записка до випускної роботи містить текстову частину, викладену на 47 сторінках друкованого тексту, має 17 сторінок додатків. Список використаних джерел містить 41 найменування і займає 5 сторінок. У роботі наведено 15 рисунків. Загальний обсяг роботи – 64 сторінки.

Метою роботи є дослідження сучасних алгоритмів розпізнавання рукописного тексту, розробка програмного забезпечення для мобільного додатку та серверної частини, що забезпечує автоматизацію процесу оцифрування текстів, і створення готового продукту, який спрощує роботу з рукописними матеріалами.

У роботі описано аналіз предметної області та існуючих аналогів, проведено дослідження сучасних методів обробки зображень і текстів, обґрунтовано вибір інструментальних засобів. Виконано проєктування та розробку програмного забезпечення, включаючи мобільний додаток і серверну частину з інтеграцією нейронної мережі. Демонстрація роботи продукту підтвердила його функціональність та ефективність. Результати розпізнавання тексту свідчать про перспективність розробки та її можливості подальшого вдосконалення.

КЛЮЧОВІ СЛОВА: РОЗПІЗНАВАННЯ ТЕКСТУ, НЕЙРОННІ МЕРЕЖІ, ШТУЧНИЙ ІНТЕЛЕКТ, МОБІЛЬНИЙ ДОДАТОК, API, PYTHON.

ABSTRACT

The master's thesis consists of a study of handwriting recognition methods, development of a mobile application for digitizing handwritten materials, creation of a server part for image processing and text recognition, and an explanatory note. The explanatory note to the graduation work contains a textual part, set out on 46 pages of printed text, and has 17 pages of appendices. The list of references includes 41 items and covers 5 pages. There are 13 figures in the work. The total volume of the work is 63 pages.

The purpose of the work is to study modern handwriting recognition algorithms, develop software for a mobile application and a server part that automates the process of digitizing texts, and create a finished product that simplifies the work with handwritten materials.

The paper describes an analysis of the subject area and existing analogs, a study of modern methods of image and text processing, and a justification for the choice of tools. The design and development of software, including a mobile application and a server side with neural network integration, was carried out. A demonstration of the product confirmed its functionality and efficiency. The results of text recognition indicate the prospects of the development and its potential for further improvement.

KEYWORDS: TEXT RECOGNITION, NEURAL NETWORKS, ARTIFICIAL INTELLIGENCE, MOBILE APPLICATION, API, PYTHON.

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ЗМІСТ	5
ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	6
ВСТУП	7
РОЗДІЛ 1. ПОШУК, АНАЛІЗ ТА ДОСЛІДЖЕННЯ ІСНУЮЧИХ МЕТОДІВ ТА ЗАСОБІВ РЕАЛІЗАЦІЇ СИСТЕМИ	10
1.1. Постановка задачі	10
1.2. Огляд й аналіз існуючих аналогічних додатків	12
1.3. Методи та алгоритми розпізнавання рукописного тексту	14
1.4. Аналіз та обґрунтування вибору інструментальних засобів	21
Висновки до першого розділу	23
РОЗДІЛ 2. ДЕТАЛЬНИЙ АНАЛІЗ ТЕХНОЛОГІЙ ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	25
2.1. Пошук та аналіз набору даних	25
2.2. Опис архітектури програмного забезпечення дослідження	27
Висновки до другого розділу	33
РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗАСТОСУВАННЯ АЛГОРИТМУ	34
3.1. Розробка мобільного застосунку	34
3.2. Розробка серверної частини	36
3.3. Демонстрація процесу роботи з додатком	38
Висновки до третього розділу	41
ВИСНОВКИ	42
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	44
ДОДАТКИ	48

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ПЗ – Програмне забезпечення

API – Application Programming Interface

CNN – Convolutional Neural Network

RNN – Recurrent Neural Network

CTC – Connectionist Temporal Classification

JSON – JavaScript Object Notation

HTTP – HyperText Transfer Protocol

ML – Machine Learning

OCR – Optical Character Recognition

HTR – Handwritten Text Recognition

ВСТУП

Актуальність теми. Сьогодні технології розпізнавання тексту стають дедалі більш актуальними та затребуваними. Від електронного документообігу до інтерфейсів користувача – розпізнавання рукописного тексту відкриває нові можливості для автоматизації процесів і підвищення ефективності роботи. З розвитком штучного інтелекту та машинного навчання, зокрема глибокого навчання, розпізнавання рукописного тексту стало значно точнішим та надійнішим.

Сучасні системи розпізнавання рукописного тексту використовують комплексні підходи, що включають обробку зображень, сегментацію, класифікацію та інші методи комп'ютерного зору і машинного навчання. Вони здатні працювати з великою кількістю даних, що отримуються з різних джерел, таких як скановані документи, фотографії, цифрові рукописи.

Поширеність рукописного тексту, який незважаючи на тенденції діджиталізації, залишається одним із найпростіших способів фіксації інформації, крім того чимало інформації вже зберігається у рукописному форматі. Розробка відповідного алгоритму та додатку націлена на вирішення задачі перетворення користувачами власних рукописів в електронний формат для подальшої роботи з цифровими даними. Популярність мобільних пристроїв, які є невід'ємною частиною повсякденного життя людей вказує на актуальність цієї платформи для реалізації застосунку розпізнавання рукописного тексту. Розробка необхідного алгоритму включає в себе використання штучного інтелекту та машинного навчання. Аналіз та покращення існуючих алгоритмів допоможе підвищити рівень точності розпізнавання та оптимізувати процес. Зростання конкуренції на ринку мобільних додатків стимулює розробників до пошуку інноваційних рішень.

Мета і завдання дослідження. Метою даного дослідження є створення мобільного додатку для розпізнавання рукописного тексту та оптимізація алгоритмів цього процесу з метою підвищення точності та продуктивності розпізнавання.

Завдання дослідження:

- аналіз існуючих методів та технологій розпізнавання рукописного тексту;
- розробка алгоритмів для розпізнавання рукописного тексту;
- оптимізація алгоритмів для забезпечення швидкодії та коректності розпізнавання;
- тестування та вдосконалення мобільного додатку з використанням реалізованої системи розпізнавання

Об'єктом дослідження є методи та підходи, використані в розробці систем розпізнавання рукописного тексту, а також набори даних, що використовуються для навчання моделей.

Предметом дослідження є методи попередньої обробки зображень, алгоритми машинного навчання для розпізнавання рукописного тексту та методи оцінки якості роботи системи, а також інструменти, які знадобляться для розробки системи.

Наукова новизна отриманих результатів полягає у тому, що запропоновано власний комбінований підхід до розпізнавання тексту, що поєднує згорткову нейронну мережу, рекурентну нейронну мережу та функцію Connectionist Temporal Classification, що дозволяє ефективно обробляти рукописний текст без попередньої сегментації символів. Створено спеціалізоване API, що реалізує зазначений алгоритм розпізнавання та надає можливість інтеграції з іншими системами чи застосунками.

Практичне значення роботи полягає у тому, що розроблено мобільний додаток для швидкого фотографування, обробки та розпізнавання текстів, що дозволяє застосовувати систему у повсякденному житті для автоматизації введення текстової інформації. Створене API може використовуватися іншими розробниками для інтеграції алгоритму розпізнавання у їхні власні програмні рішення. Напрацьований матеріал і результати дослідження можуть бути використані для подальших розробок у сфері комп'ютерного зору та розпізнавання текстів. результати роботи можуть бути застосовані у створенні

автоматизованих систем, в яких необхідна автоматична обробка зображень текстів.

Апробація. За темою кваліфікаційної магістерської роботи було опубліковано 2 тези наукових конференцій:

- Леут Назарій., Вакалюк Тетяна. Мобільний додаток для розпізнавання рукописного тексту / Збірник матеріалів XI Всеукраїнської науково-практичної конференції молодих вчених «Наукова молодь-2023» (Київ, 21 листопада 2023 р.). / упоряд.: А. Яцишин. К.: ЦП «КОМПРИНТ», 2023. – С. 105 – 106 [1];
- Leut N. THE USE OF RECURRENT NEURAL NETWORKS FOR HANDWRITTEN TEXT RECOGNITION / Збірник тез доповідей II Міжнар. наук.-практич. конфер. «Інновації та перспективні шляхи розвитку інформаційних технологій» (06 груд. 2023 р., м. Черкаси) [Електронний ресурс] / упоряд. : Т. О. Прокопенко, Я. В. Тарасенко ; М-во освіти і науки України, Черкас. держ. технол. ун-т. – Черкаси : ЧДТУ, 2023. – С. 217 – 218 [2].

РОЗДІЛ 1. ПОШУК, АНАЛІЗ ТА ДОСЛІДЖЕННЯ ІСНУЮЧИХ МЕТОДІВ ТА ЗАСОБІВ РЕАЛІЗАЦІЇ СИСТЕМИ

1.1. Постановка задачі

В сучасному світі мобільні додатки визначають стандарти зручності та ефективності в повсякденному житті. Зокрема, розробка додатків для розпізнавання рукописного тексту стає невід’ємною ланкою цього невпинного технологічного розвитку. Незважаючи на прогресуючу цифровізацію, рукописний текст залишається популярним методом фіксації інформації, а відтак, виникає потреба в зручних та ефективних інструментах для його розпізнавання та обробки.

Мобільні застосунки для розпізнавання рукописного тексту вирішують проблеми користувачів, дозволяючи легко конвертувати нотатки, записи або рукописні документи у цифровий формат. Це надає можливість подальшого редагування, швидкого пошуку та зручного зберігання важливої інформації. Додатки відзначаються високою ефективністю, навіть у випадках нечіткого написання, і надають користувачам різні опції перетворення рукописного тексту, такі як переклад чи конвертація в інші формати. Забезпечення зрозумілого та дружнього інтерфейсу робить використання додатків максимально зручним та доступним [3].

В основі роботи таких додатків лежать потужні алгоритми розпізнавання символів та слів. Деякі з них можуть навіть адаптуватися до стилю рукопису конкретного користувача, стаючи все точнішими у розпізнаванні тексту з плином часу. Це підвищує якість та швидкість роботи додатків, роблячи їх необхідним інструментом для ефективної обробки рукописної інформації в цифровому форматі [4].

Найпоширенішими моделями розпізнавання тексту є OCR (Optical Character Recognition – оптичне розпізнавання символів) та HTR (Handwritten Text Recognition – розпізнавання рукописного тексту). Відзначаючись високою точністю, OCR вирішує завдання розпізнавання друкованих символів

без надмірних обчислень. З іншого боку, НТР вимагає застосування штучних нейронних мереж та машинного навчання, відкриваючи нові горизонти для розпізнавання рукописного тексту.

Недавні прориви в галузі штучних нейронних мереж (ШНМ) суттєво трансформують парадигму розпізнавання рукописного тексту. У порівнянні з традиційними методами, ШНМ автоматично навчаються на прикладах, що дозволяє їм адаптуватися до різноманітних варіацій рукопису. Зокрема, згорткові нейронні мережі стали ключовим інструментом для задач розпізнавання зображень, включаючи рукописний текст.

Згорткові нейронні мережі (ЗНМ) володіють здатністю автоматично виділяти важливі ознаки з вхідних зображень, використовуючи фільтри, які адаптуються під час навчання. Ці мережі структуровані так, що вони здатні розпізнавати ієрархічні ознаки на різних рівнях абстракції. Застосовані до розпізнавання рукописного тексту, ЗНМ здатні ефективно впоратися з різноманітністю стилів написання та вирізняються високою точністю. За останні роки технологія згорткових нейронних мереж в суттєвий спосіб розвинулася, надаючи практичне застосування в різних галузях, включаючи розпізнавання рукописного тексту [5].

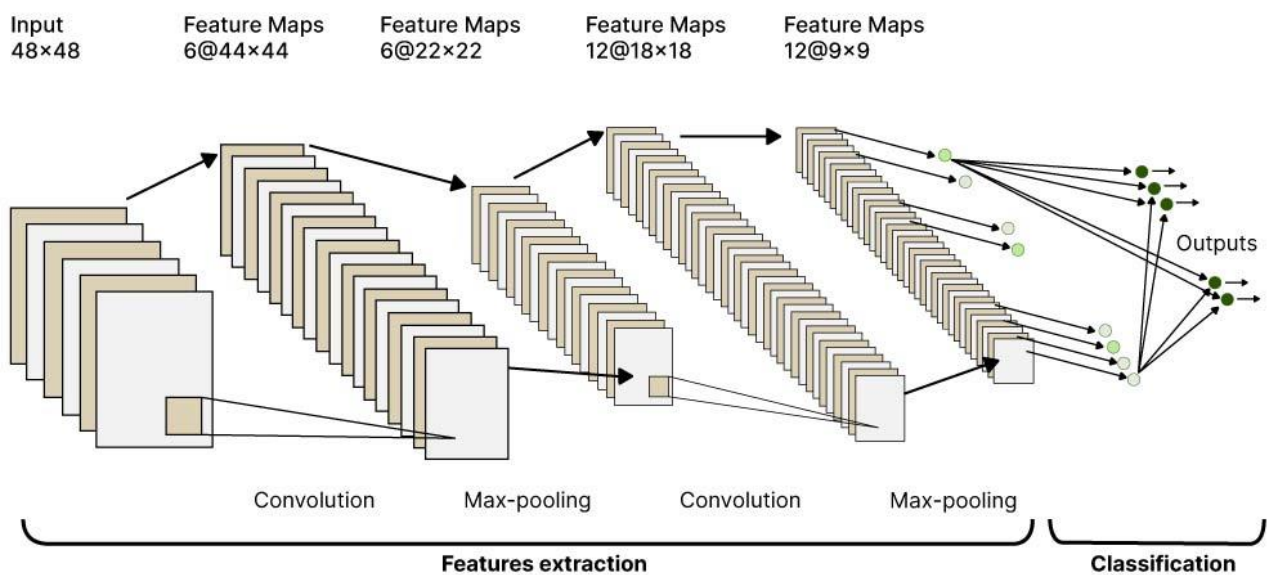


Рис. 1.1. Структура згорткової мережі

Головною метою є реалізація власного підходу до розпізнавання рукописного тексту та демонстрація його роботи в розробленому мобільному

застосунку. Детально оглянувши основні принципи найбільш ефективних існуючих систем, поставимо перед собою такі задачі, як аналіз аналогічних рішень, виділення сильних і слабких сторін алгоритмів, що використовуються для НТР та на основі отриманих висновків проєктування та реалізація алгоритму розпізнавання, розробка клієнтської частини у форматі мобільного застосунку та імплементація алгоритму в нього.

1.2. Огляд й аналіз існуючих аналогічних додатків

Сучасний швидкоплинний розвиток технологій неухильно визначає тенденції у сфері програмного забезпечення. Розглядаючи питання створення нового додатка, важливо докладно вивчити та проаналізувати існуючі аналогічні рішення на ринку. Аналіз конкурентів допоможе виявити можливості покращення нового продукту та розкрити їхні сильні та слабкі сторони, що визначить стратегію розвитку та допоможе позиціонувати новий додаток ефективним чином. Для порівняння було розглянуто такі додатки:

- Microsoft OneNote [6];
- Adobe Scan [7].

Microsoft OneNote – це розроблена корпорацією Microsoft програма для створення, організації та зберігання нотаток. Вона дозволяє користувачам створювати цифрові записи, робити нотатки, вставляти мультимедійний вміст, організовувати інформацію. На рис. 1.2 зображено інтерфейс програми.

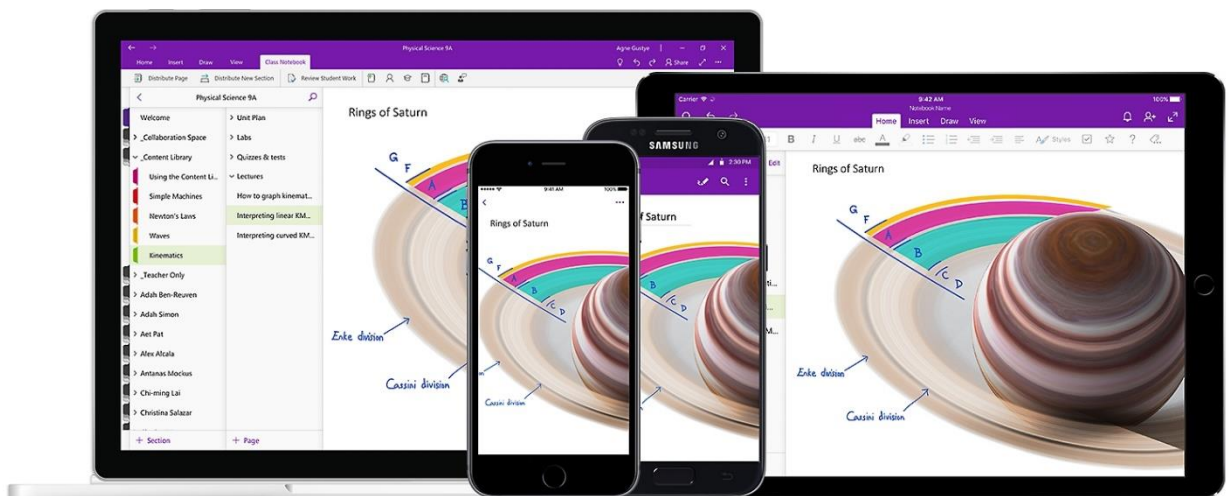


Рис. 1.2. Інтерфейс Microsoft OneNote

Цей інструмент пропонує широкий функціонал для роботи з текстовими та мультимедійними матеріалами. У Microsoft OneNote можна створювати записи, впорядковувати інформацію за тематичними розділами, додавати аудіо- та відеофайли, а також вставляти рукописні примітки. Однією з особливостей програми є її інтеграція з іншими продуктами Microsoft – Office, Outlook та OneDrive, що дозволяє швидко обмінюватися інформацією та працювати над документами у команді. OneNote доступний більшості популярних операційних систем – Windows, macOS, iOS та Android, крім того він має веб-версію. Це забезпечує гнучкість у використанні. Попри зазначені переваги, інтерфейс програми може викликати труднощі у користувачів через його перевантаженість великою кількістю функцій та інструментів. Також додаток не пристосований для інтеграції з сервісами поза екосистемою Microsoft.

Adobe Scan – це мобільний додаток, розроблений компанією Adobe, який дозволяє користувачам здійснювати сканування документів за допомогою камери смартфона чи планшета. Він використовує технологію оптичного розпізнавання символів, завдяки чому користувачі можуть отримувати текстові документи зі зроблених фотографій. Програма також надає можливість редагування сканованих матеріалів. Збереження отриманих файлів доступне у хмарних сховищах Adobe Document Cloud, Google Drive та Dropbox. Базовий функціонал додатку є безкоштовним, що робить його доступним для широкого кола користувачів, однак більшість інструментів доступні у платній версії, а інтеграція з іншими сервісами обмежена. На рис. 1.3. зображено інтерфейс додатку.

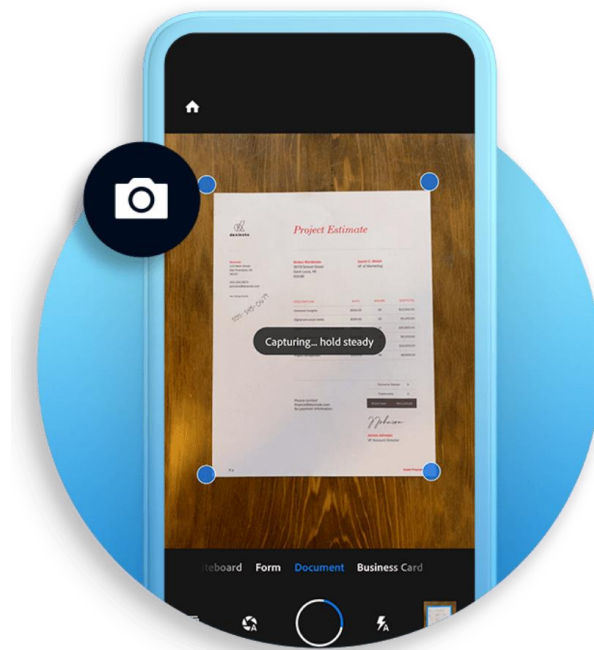


Рис. 1.3. Інтерфейс Adobe Scan

Грунтуючись на результатах аналізу сформуємо власні вимоги до програмного продукту, що розроблятиметься. Перш за все – ключовим елементом є ефективний алгоритм розпізнавання саме рукописного а не друкованого тексту, крім того зручний та простий у користуванні інтерфейс користувача забезпечить найкращий досвід роботи з застосунком. Сфокусуємося на основних функціях – скануванні та копіюванні розпізнаного тексту для подальшого використання. Зробивши продукт безкоштовним та з відкритим кодом, ми можемо залучити спільноту до подальших інтеграцій чи вдосконалень як проєкту загалом, так і його частин.

1.3. Методи та алгоритми розпізнавання рукописного тексту

Розпізнавання рукописного тексту є складною задачею, що включає декілька етапів [8], таких як сегментація, вилучення ознак, класифікація та постобробка. На кожному з цих етапів використовуються різноманітні методи та алгоритми, які спрямовані на досягнення високої точності розпізнавання. Розглянемо основні підходи до розпізнавання рукописного тексту, зокрема традиційні методи, методи на основі машинного навчання та глибокого навчання, принцип їх дії, переваги та недоліки кожного з них.

Шаблонне розпізнавання [9] (Template Matching) є одним з найпростіших і базових методів для ідентифікації рукописного тексту. Його принцип роботи полягає в тому, що кожен символ на зображенні порівнюється зі зразками – шаблонами, які заздалегідь збережені в базі даних. При цьому здійснюється пошук найбільш схожого зразка на основі певного методу порівняння, наприклад, кореляції пікселів або мінімізації відстані між зображеннями. Цей підхід широко застосовується для задач, де набір символів обмежений або текст має чітко визначені характеристики, як, наприклад, друковані шрифти.

Головна перевага цього методу полягає у простоті реалізації. Оскільки алгоритм не вимагає складної обробки даних або навчання, його можна швидко впровадити навіть на пристроях із обмеженими обчислювальними ресурсами. Крім того, метод забезпечує високу швидкість обробки, якщо кількість шаблонів у базі даних є невеликою. Це робить його корисним у випадках, коли потрібно розпізнати текст із фіксованим набором символів, наприклад, при роботі з друкованими текстами або стандартними формами рукопису.

Однак зіставлення з шаблоном має значні обмеження, які зменшують його ефективність у більшості задач розпізнавання. Ключовим недоліком є низька варіативність у написанні символів. Рукописний текст різних людей може суттєво відрізнятися за формою, розміром, нахилом і товщиною ліній. Для забезпечення коректного розпізнавання кожного можливого варіанта символу необхідно зберігати великий набір шаблонів, що значно збільшує обсяг бази даних і ускладнює сам процес порівняння. Якість самого розпізнавання сильно залежить від якості вихідного зображення. Шум, розмиття або нерівномірне освітлення суттєво впливають на точність порівняння.

Методи на основі виділення ознак [10][11] (Feature Extraction Methods) є більш складним та ефективним підходом до розпізнавання рукописного тексту. Їх принцип роботи полягає у виділенні ключових характеристик

символів, таких як контури, кути, структури або текстурні особливості. Після цього ці характеристики використовуються як вхідні дані для класифікації символів. Для цього залучаються методи машинного навчання або статистичного аналізу. Завдяки виділенню інформативних ознак зображення стає менш залежним від таких факторів, як нахил або розмір символів, що підвищує точність розпізнавання.

Однією з головних переваг цього підходу є його стійкість до варіацій у написанні символів. Завдяки цьому методи на основі виділення ознак значно перевершують шаблонне розпізнавання у випадках, коли символи написані різними почерками або мають незначні деформації. Використання різних типів ознак, дозволяє адаптувати алгоритми для конкретних задач і покращувати точність розпізнавання навіть у складних умовах.

Складність у виборі та комбінуванні правильних ознак та пошук оптимального набору характеристик вимагає значного досвіду та експериментальної роботи, це вимагає виділення значних ресурсів тому є головним недоліком.

Метод k-найближчих сусідів [11][12] (k-Nearest Neighbors, k-NN) ґрунтується на класифікації нового зразка на основі його схожості з іншими зразками з навчальної вибірки. У просторі ознак, який використовується для представлення даних, обчислюється відстань між зразками. Для нового символу визначаються k найближчих сусідів за певною метрикою відстані. Клас символу встановлюється на основі більшості класів серед його найближчих сусідів.

Основною перевагою методу k-NN є його простота у реалізації та зрозумілість. Алгоритм не потребує складного етапу навчання, оскільки всі операції класифікації виконуються під час запиту. Гнучкість у виборі функції відстані та кількості сусідів дозволяє адаптувати метод до різних задач і умов розпізнавання. Для якісніших вхідних даних можна використовувати невелике значення параметра k, а для даних із високим рівнем шуму – більші значення k, що допомагає уникнути помилок через одиничні аномалії.

Основним недоліком цього методу є високі обчислювальні витрати. Оскільки для кожного нового зразка потрібно обчислювати відстань до всіх зразків навчальної вибірки, це є серйозною проблемою при роботі зі значними обсягами даних. Чутливість до вибору параметра k також є важливою проблемою: занадто мале значення може призвести до перенавчання, тоді як занадто велике – до втрати важливої локальної інформації. Ефективність методу може суттєво знижуватися в умовах високої вимірності простору ознак.

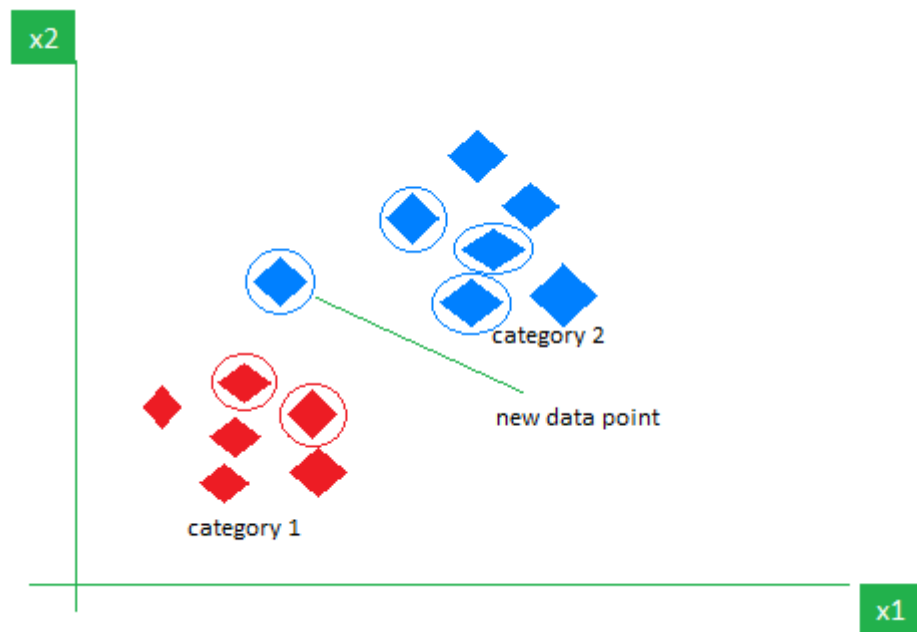


Рис. 1.4. Принцип роботи k-NN

Метод опорних векторів [12] (Support Vector Machines, SVM) знаходить гіперплощину, яка максимально розділяє класи в багатовимірному просторі ознак. Для цього використовується поняття маржі – відстані між гіперплощиною та найближчими точками кожного класу. Необхідно знайти таку гіперплощину, яка забезпечує максимальну маржу, що мінімізує ризик помилок класифікації. У випадках, коли дані не можуть бути лінійно розділені в початковому просторі, алгоритм використовує спеціальні функції ядра (kernel functions), які проектують дані в простір більшої розмірності. У цьому просторі класи стають лінійно роздільними, що дозволяє застосовувати стандартний підхід.

Метод опорних векторів дуже ефективний для задач із малими та середніми наборами даних, оскільки використовує лише дані, що лежать на межі між класами для побудови гіперплощини. Для цього процесу не знадобиться великої кількості ресурсів. SVM добре працює з багатовимірними даними, що робить його універсальним інструментом для різноманітних задач класифікації, стійкий до перенавчання за умови вдалого налаштування параметрів. Однак висока вимогливість до обчислювальних ресурсів при роботі з великими наборами даних є суттєвим недоліком, оскільки обчислення відстаней і побудова гіперплощини потребує значних ресурсів. Крім того, вибір правильного ядра та його параметрів вимагає експериментального підходу. Ще однією проблемою є чутливість до шуму в даних.

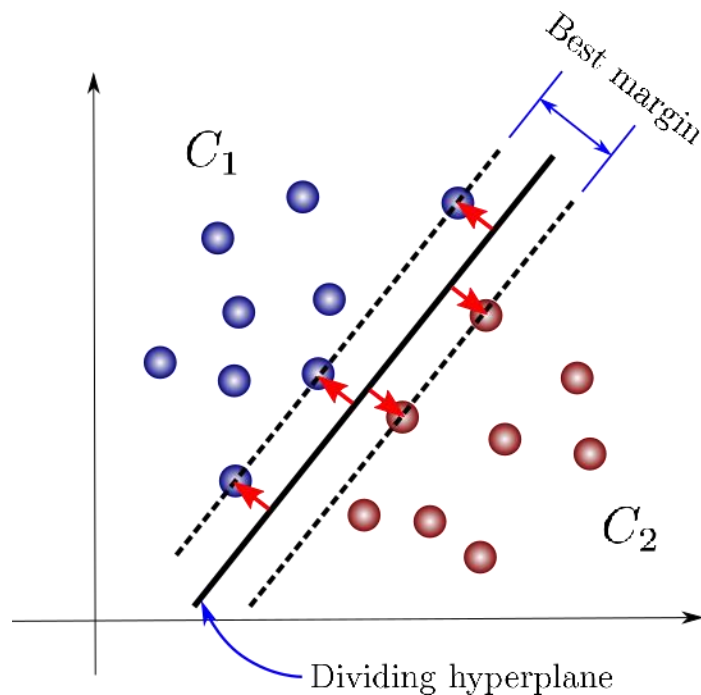


Рис. 1.5. Принцип роботи SVM

Нейронні мережі [13] (Neural Networks) складаються з великої кількості взаємопов'язаних нейронів, що утворюють шари. Процес обробки інформації полягає у проходженні вхідних даних через нейрони, кожен із яких застосовує вагові коефіцієнти, виконує обчислення за допомогою активаційної функції та передає результат на наступний шар. Особливістю нейронних мереж є їх здатність автоматично виділяти ознаки з вхідних даних без необхідності їх ручного визначення.

Нейронні мережі універсальні і здатні адаптуватися до варіацій у вхідних даних. Вони демонструють високу точність на великих наборах даних, ефективно знаходячи складні закономірності. Недоліками цього підходу є висока обчислювальна складність і необхідність великої кількості даних для навчання, підбір параметрів потребує досвіду чи експериментування.

Згорткові нейронні мережі [14][15] (Convolutional Neural Networks, CNN) є спеціалізованим типом нейронних мереж, які широко використовуються для обробки зображень. Основним елементом їхньої архітектури є шари згортки, які виконують автоматичне виділення ознак зображення. Згорткові шари застосовують фільтри, що проходять через зображення, дозволяючи виявляти локальні ознаки, такі як контури, кути, текстури. Після шарів згортки слідує шар об'єднання (pooling), метою яких є зменшення розмірності даних, що зменшує кількість параметрів і обчислювальну складність, робить модель стійкішою до варіацій, таких як зміщення або масштабування символів. В кінці мережі зазвичай використовуються повнозв'язні шари, які виконують класифікацію на основі ознак, виділених попередніми шарами. Використання шарів підсумовування зменшує ризик перенавчання за рахунок зменшення розмірності даних. Принцип роботи RNN зображено на рис. 1.1.

Серед недоліків – CNN вимагають значних обчислювальних ресурсів і пам'яті для навчання. Налаштування архітектури, вибір кількості і розмірів фільтрів, типів активаційних функцій та параметрів навчання є складним і вимагає експериментального підходу. Згорткові нейронні мережі ефективно працюють лише за умови наявності великої кількості навчальних даних.

Рекурентні нейронні мережі [16][17] (Recurrent Neural Networks, RNN) використовуються для обробки послідовних даних, зокрема тексту. Головною особливістю цих мереж є наявність зворотних зв'язків, які дозволяють зберігати інформацію про попередні символи в послідовності. Завдяки цьому RNN можуть моделювати залежності між елементами послідовності. Модифікації, такі як LSTM (Long Short-Term Memory) і GRU (Gated Recurrent

Unit) вирішують проблему зникнення градієнта і дозволяють зберігати довготривалі залежності та покращувати ефективність навчання.

Високі вимоги до обчислювальних ресурсів є основним недоліком RNN. Крім того, налаштування і тренування таких моделей є складним завданням, що вимагає ретельного вибору параметрів і архітектури. Ще одним обмеженням є потреба в значному обсязі навчальних даних для досягнення високої точності, що може бути проблематичним у задачах із обмеженим доступом до великих датасетів.

Було вирішено використати комбінацію згорткової нейронної мережі та рекурентної нейронної мережі разом з функцією Connectionist Temporal Classification (CTC) [18]. Такий підхід обирається через його високу ефективність у задачах розпізнавання тексту, особливо коли структура даних є послідовною і містить часові залежності. Крім цього необхідно знайти великий підходящий набір даних, оскільки обрані методи потребують великої кількості навчальних даних.

CNN здатні автоматично виявляти значущі ознаки на різних рівнях абстракції з вхідного зображення. Шари згортки обробляють локальні патерни, що особливо важливо для обробки зображень символів. Шари агрегування зменшують розмірність простору ознак, що зменшує обчислювальну складність. RNN зберігають інформацію про попередні символи, що важливо для розпізнавання послідовного тексту. Connectionist Temporal Classification (CTC) дозволяє моделі навчатись без потреби в попередній сегментації символів, що значно спрощує процес підготовки даних.

CTC є функцією втрат (loss function), розробленою для навчання моделей, які генерують послідовності з невизначеною довжиною. Вона застосовується в задачах, де вхідні дані мають одну довжину, а вихідні послідовності можуть мати іншу, як, у нашому випадку, розпізнавання рукописного тексту. CTC знаходить усі можливі відповідності між вхідною послідовністю (зображенням тексту) та вихідною послідовністю (розпізнаним

текстом). Це дозволяє моделі враховувати різні можливі вирівнювання між вхідними та вихідними символами. Вводиться спеціальний символ «blank», який представляє відсутність символу на даній позиції. Це допомагає моделі коректно розпізнавати випадки, коли деякі позиції у вхідній послідовності не відповідають жодному символу у вихідній. Для кожного можливого вирівнювання між вхідними та вихідними символами обчислюється ймовірність. Ймовірності всіх можливих вирівнювань сумуються, щоб отримати кінцеву ймовірність відповідності вхідної та вихідної послідовностей після чого використовується негативний логарифм ймовірності правильного вирівнювання як функцію втрат. Метою навчання моделі є мінімізація цієї функції втрат.

1.4. Аналіз та обґрунтування вибору інструментальних засобів

Проаналізувавши вимоги майбутнього додатку, було вирішено розробляти додаток з використанням такого популярного підходу як RESTful API [19].

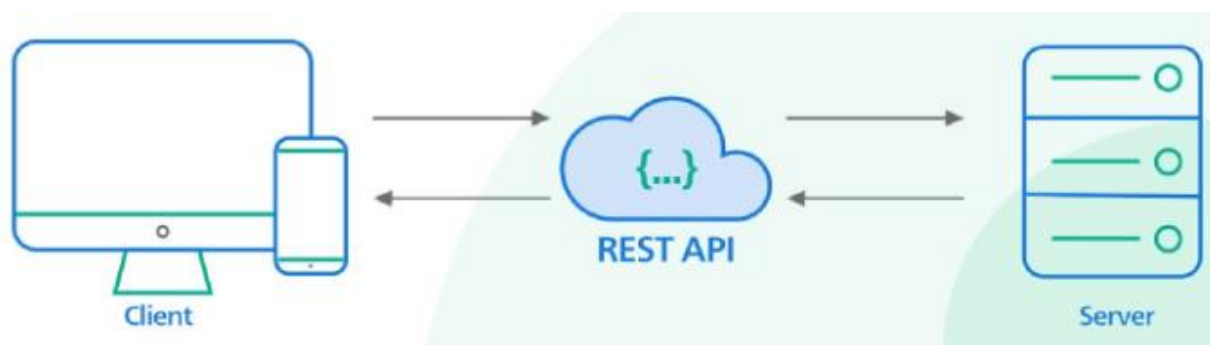


Рисунок 1.7 – Загальний принцип роботи RESTful API

Однією з головних характеристик є розмежування обов'язків між клієнтом і сервером. Це полегшує інтеграцію різних систем, зменшує складність розробки та сприяє швидшому впровадженню рішень. Універсальність RESTful API забезпечується його незалежністю від конкретних мов програмування та платформ, тому клієнт та сервер можуть використовувати різні технологічні стеки. Використання HTTP-протоколу забезпечує дотримання усталених веб-стандартів, зокрема HTTPS дозволяє захистити дані, що передаються між клієнтом і сервером. RESTful API

дозволяє легко змінювати структуру або поведінку серверу без необхідності значних змін на стороні клієнтів, що в свою чергу полегшить можливі інтеграції в майбутньому. Тестування та документування RESTful API є простими завдяки його стандартизованій природі, що значно полегшує процес налагодження системи та її передачу до використання.

Для реалізації серверної частини було обрано мову програмування Python [20], адже Python має велику кількість бібліотек та фреймворків, які допомагають у різноманітних завданнях, включаючи обробку даних, машинне навчання (наприклад pandas [21] – програмна бібліотека для маніпулювання даними та їхнього аналізу). Python має велику та активну спільноту розробників, яка створює та підтримує різні інструменти для розробки та оптимізації нейронних мереж. Існує багато бібліотек для обробки та аналізу зображень в Python, таких як OpenCV [22] та scikit-image [23] які дозволяють виконувати операції з редагування, сегментації та аналізу зображень. Це важливо для попередньої обробки та підготовки зображень перед введенням їх до нейронної мережі. Також наявні фреймворки для реалізації серверної частини додатку, зокрема FastAPI [24] який буде використано при написанні серверної частини додатку. Python є мовою загального призначення та легко інтегрується з іншими технологіями та сервісами, що є надзвичайно важливим для розширення функціональності системи розпізнавання рукописного тексту в майбутньому.

Для реалізації мобільного додатку було вирішено сфокусуватися на одній платформі, тому обрано було найпопулярнішою операційну систему – Android. У майбутньому можливе розширення до інших систем. Було визначено використання Kotlin [25] та Android Studio [26]. Kotlin є офіційно підтримуваною мовою для розробки під Android, що забезпечує доступ до останніх оновлень та інструментів екосистеми. Додаток легко інтегрується з будь-яким RESTful API. Середовищем розробки було обрано Android Studio, що надає зручний інтерфейс та всі необхідні нам інструменти.

Для реалізації API було обрано FastAPI, оскільки фреймворк дозволяє швидко розгорнути сервери з мінімальним кодом, завдяки зручному і зрозумілому синтаксису, асинхронна природа FastAPI дозволяє ефективно обробляти кілька запитів одночасно. Також FastAPI автоматично генерує OpenAPI-документацію (Swagger UI), що спрощує тестування API і робить його зручним для інтеграції з іншими системами. Використання Python дозволяє легко інтегрувати навчену нейронну модель в серверну частину.

Вимоги до мобільних пристроїв:

- Android 7+
- з'єднання з мережею Internet

Вимоги до серверного обладнання:

- процесор з частотою від 3 GHz
- мінімум 2ГБ оперативної пам'яті
- Python 3.10+

Окрім описаного інструментарію, можливе використання додаткових сторонніх бібліотек, які доповнюють вказані та додають функціонал або надають інші моделі машинного навчання до використання.

Висновки до першого розділу

У рамках першого розділу було визначено основні задачі для подальшого дослідження та розробки мобільного додатку для розпізнавання рукописного тексту. На основі аналізу предметної області та завдань, які вирішує проєкт, було сформовано чіткі вимоги до функціоналу програми.

Протягом проведення огляду сучасних аналогів, таких як Microsoft OneNote та Adobe Scan, було проаналізовано їхній функціонал, сильні та слабкі сторони. Завдяки цьому вдалося виділити ключові аспекти, які потребують вдосконалення або розширення у розробленому продукті. Зокрема, це точність розпізнавання тексту, зручність інтерфейсу, швидкість обробки та рівень конфіденційності даних. Цей аналіз сформував основу для визначення орієнтирів у реалізації власної системи.

Дослідження методів та алгоритмів розпізнавання рукописного тексту дозволило обрати сучасні підходи, які забезпечують високу ефективність та точність роботи системи. Зокрема, використання комбінації згорткових і рекурентних нейронних мереж у поєднанні з функцією Connectionist Temporal Classification (CTC) було визначено як оптимальний підхід для вирішення задачі розпізнавання рукописного тексту.

Аналіз інструментальних засобів для розробки мобільного додатку та серверної частини дозволив обґрунтовано обрати Kotlin та Android Studio для реалізації клієнтської частини, а також FastAPI для створення серверної частини. Ці інструменти забезпечують оптимальний баланс між швидкістю розробки, продуктивністю та зручністю масштабування.

В результаті проведеного аналізу предметної області, огляду аналогічних рішень, методів розпізнавання тексту та інструментальних засобів було сформовано концепцію розробки додатку, що дозволяє ефективно вирішити поставлену задачу та задовольнити потреби цільової аудиторії.

РОЗДІЛ 2. ДЕТАЛЬНИЙ АНАЛІЗ ТЕХНОЛОГІЙ ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Пошук та аналіз набору даних

Для розробки системи розпізнавання рукописного тексту було використано набір даних IAM Sentences [27], який містить зображення тексту речень, написаних від руки, з відповідними текстовими мітками. Цей набір даних є доволі популярним у галузі і перевірений часом, часто використовується для оцінки ефективності моделей розпізнавання тексту.

Перед початком навчання моделі необхідно провести аналіз та попередню обробку даних. Кожен зразок у цьому наборі даних складається із зображення рукописного тексту та відповідного зображенню рядка тексту. Файл sentences.txt містить метадані для кожного зразка зображення в наборі даних, включаючи шлях до файлу зображення та відповідну текстову мітку (рис. 2.1).

```

1  #--- sentences.txt -----#
2  #
3  # iam database sentence information
4  #
5  # format: a01-000u-s0-00 0 ok 154 19 408 746 1663 91 A|MOVE|to|stop|Mr.|Gaitskell|from
6  #
7  # a01-000u-s0-00 -> sentence/line id for form a01-000u
8  # 0 -> sentence number within this form
9  # ok -> result of word segmentation
10 # ok: line is correctly segmented
11 # er: segmentation of line has one or more errors
12 #
13 # warning: if a sentence starts or ends in the middle of
14 # a line which is not correctly segmented, a
15 # correct extraction of the sentence can fail.
16 #
17 # 154 -> graylevel to binarize line
18 # 19 -> number of components for this part of the sentence
19 # 408 746 1663 91 -> bounding box around for this part of the sentence
20 # in the x,y,w,h format
21 #
22 # A|MOVE|to|stop|Mr.|Gaitskell|from
23 # -> transcription for this part of the sentence. word
24 # tokens are separated by the character |
25 #
26 a01-000u-s00-00 0 ok 154 19 408 746 1661 91 A|MOVE|to|stop|Mr.|Gaitskell|from
27 a01-000u-s00-01 0 ok 156 19 395 932 1850 105 nominating|any|more|Labour|life|Peers
28 a01-000u-s00-02 0 ok 157 16 408 1106 1986 105 is|to|be|made|at|a|meeting|of|Labour

```

Рис. 2.1. Структура набору даних

Створимо змінні dataset, кожен елемент якого міститиме шлях до файлу зображення та текстову мітку, що дозволяє легко звертатися до даних під час

навчання моделі, vocab – набір, що міститиме унікальні символи міток та max_length – найбільшу довжину міток:

```
dataset, vocab, max_length = [], set(), 0
```

Розглянемо кожен рядок файлу sentences.txt. Якщо рядок починається з «#» - пропускаємо його:

```
words = open(sentences_txt_path, "r").readlines()

for line in tqdm(words):
    if line.startswith("#"):
        continue
```

Після цього отримуємо елементи рядка розділені пробілами та розглядаємо далі, якщо третій елемент не «err» (в наборі помічені дані):

```
line_split = line.split(" ")
if line_split[2] == "err":
    continue
```

Отримуємо назву файлів, взявши перший елемент рядка:

```
file_name = line_split[0] + ".png"
```

Замінімо всі символи «|» на пробіли, для отримання тексту рядків:

```
label = line_split[-1].rstrip("\n").replace("|", " ")
```

Створюємо змінну шляху до файлу. Якщо шляху до файлу не існує – пропускаємо цей рядок:

```
rel_path = os.path.join(sentences_folder_path, file_name)
if not os.path.exists(rel_path):
    print(f"Файл не знайдено: {rel_path}")
    continue
```

Якщо такий шлях є – додаємо шлях до dataset та мітку до списку наборів даних, оновлюємо змінну vocab символами з мітки та присвоюємо max_length максимальне значення між поточним значенням змінної та довжиною мітки:

```
dataset.append([rel_path, label])
vocab.update(list(label))
max_length = max(max_length, len(label))
```

Для навчання моделі необхідно мати набір усіх унікальних символів, які зустрічаються у текстових мітках, що дозволяє моделі працювати з обмеженим набором символів і спрощує процес навчання. Максимальна довжина мітки використовується для фіксування розміру вихідного масиву, що генерується моделлю і потрібна для коректної роботи алгоритму Connectionist Temporal Classification.

2.2. Опис архітектури програмного забезпечення дослідження

Після попередньої обробки даних наступним кроком є створення та навчання моделі розпізнавання рукописного тексту. Спершу заповнимо конфігураційний файл `configs.py` (див. додаток А) та створимо об'єкт `ModelConfigs` на його основі для зберігання конфігурацій моделі, задамо словник `vocab` і максимальну довжину `max_length` тексту як атрибути цього об'єкта. Цей конфігураційний файл знадобиться при запуску виведення на навченій моделі:

```
configs = ModelConfigs()
configs.vocab = "".join(sorted(vocab))
configs.max_text_length = max_length
configs.save()
```

Потім розділимо набір даних у співвідношенні 90% для навчання і 10% для перевірки.

```
data_df = pd.DataFrame(dataset, columns=["image_path", "label"])
train_df, val_df = train_test_split(data_df, test_size=0.2, random_state=42)

train_df.to_csv(os.path.join(configs.model_path, "train.csv"), index=False)
val_df.to_csv(os.path.join(configs.model_path, "val.csv"), index=False)
```

Крім цього створимо об'єкти застосуємо випадкові яскравість, роздільну здатність та різкість для поповнення даних.

```
train_data_gen = ImageDataGenerator(
    brightness_range=[0.8, 1.2],
    zoom_range=0.2,
    preprocessing_function=lambda x: cv2.GaussianBlur(x, (5, 5), 0)
)
```

Завантажемо та підготуємо для використання у тренуванні зображення:

```
train_images, train_labels = [], []

for index, row in train_df.iterrows():
    image_path, label = row["image_path"], row["label"]
    try:
        image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
        if image is None:
            print(f"Could not read image: {image_path}")
            continue

        image = cv2.resize(image, (configs.width, configs.height))
        image = image / 255.0 # Нормалізація

        train_images.append(np.expand_dims(image, axis=-1))
        train_labels.append(text_to_sequence(label))

    except Exception as e:
        print(f"Error processing image {image_path}: {e}")

train_images = np.array(train_images)
train_labels = np.array(train_labels)
```

Перетворимо текстові мітки у числовий формат з падингом для використання функції втрат далі.

```
# Перетворення текстових міток у числовий формат із падингом
train_labels = [text_to_sequence(label) for label in train_df["label"]]
val_labels = [text_to_sequence(label) for label in val_df["label"]]

# Застосування падингу
train_labels = pad_sequences(train_labels, maxlen=configs.max_text_length,
padding="post", value=-1)
val_labels = pad_sequences(val_labels, maxlen=configs.max_text_length,
padding="post", value=-1)

# Підготовка наборів даних
train_dataset = tf.data.Dataset.from_tensor_slices((train_images,
train_labels))
train_dataset =
train_dataset.batch(configs.batch_size).prefetch(tf.data.AUTOTUNE)

val_dataset = tf.data.Dataset.from_tensor_slices((val_images, val_labels))
val_dataset =
val_dataset.batch(configs.batch_size).prefetch(tf.data.AUTOTUNE)
```

Модель розпізнавання рукописного тексту базується на згортковій нейронній мережі. Модель отримує зображення на вході і створює послідовність міток (символьних позначок) для кожного зображення на виході.

Шари згортки використовуються для виділення ознак із зображень. LSTM-шари (long short-term memory) [28] фіксують зв'язки між символами в мітках, що дозволяє краще розуміти контекст та порядок символів. Вихідні дані з шарів LSTM проходять через щільний шар з активацією softmax [29], який створює розподіл ймовірностей для символів у словнику для кожного часового кроку, що дозволяє передбачити правильну мітку для кожного символу на вхідному зображенні. Програмний код файлу model.py наведений у додатку А.

Процес навчання моделі базується на використанні функції втрат СТС, яка дозволяє моделі прогнозувати послідовності символів без явного вирівнювання між входом і виходом. Цей підхід є ключовим для обробки даних, де довжина вхідних і вихідних послідовностей може відрізнятись.

Оптимізація параметрів здійснюється за допомогою алгоритму Adam [30], який автоматично налаштовує швидкість навчання, використовуючи

градієнти першого та другого порядків. Це забезпечує ефективну збіжність та покращує стабільність навчання.

Оцінка ефективності моделі виконується за допомогою метрик CWER[31] та WER[31]. CWER визначає відсоток помилок на рівні символів, а WER оцінює точність розпізнавання на рівні слів у реченнях. Комбінація цих метрик дозволяє точно виміряти продуктивність моделі як на символічному, так і на словесному рівнях.

Визначимо зворотні виклики, які будуть використовуватися під час навчання:

```
callbacks = [
    EarlyStopping(patience=10, restore_best_weights=True),
    ModelCheckpoint(os.path.join(configs.model_path, "best_model.h5"),
save_best_only=True),
    ReduceLROnPlateau(patience=5, factor=0.5, verbose=1),
    TensorBoard(log_dir=os.path.join(configs.model_path, "logs")),
    MetricsLogger(val_data=val_dataset, char_list=configs.vocab)
]
```

`EarlyStopping` [32] – зупиняє навчання, якщо продуктивність на валідації не покращується протягом 10 епох (`patience=10`). Параметр `restore_best_weights=True` відновлює ваги з найкращої епохи;

`ModelCheckpoint` [32] – зберігає ваги моделі у файл наприкінці кожної епохи в разі покращення продуктивності на тестовому наборі;

`ReduceLROnPlateau` [32] – зменшує швидкість навчання оптимізатора вдвічі, якщо продуктивність моделі на валідаційному наборі не покращується протягом заданих 5 епох. Повідомлення про зміну швидкості виводиться в консоль;

`TensorBoard` [33] – записує метрики навчання та валідації в лог-файли, які потім можна переглянути за допомогою веб-інструмента `TensorBoard` для візуалізації та аналізу;

`MetricsLogger` – вимірює та виводить CWER та WER під час тренування моделі для оцінки її продуктивності, допомагає контролювати якість моделі під час тренувального процесу.

Після навчання модель зберігається для подальшого використання.

Тренування моделі:

```

model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=configs.train_epochs,
    callbacks=callbacks
)

```

Процес тренування можна побачити у терміналі під час запуску програми (рис. 2.2).

```

Run: train
5426/5426 [=====] - 1234s 227ms/step - loss: 1.7455 - CER: 0.1091 - WER: 0.2863 - val_loss: 1.3524 - val_CER: 0.0855 - val_WER: 0.2209 - lr: 5.0000e-04
Epoch 63/1000
5426/5426 [=====] - ETA: 0s - loss: 1.7319 - CER: 0.1090 - WER: 0.2843
5426/5426 [=====] - 1220s 225ms/step - loss: 1.7319 - CER: 0.1090 - WER: 0.2843 - val_loss: 1.3642 - val_CER: 0.0852 - val_WER: 0.2241 - lr: 5.0000e-04
Epoch 64/1000
5426/5426 [=====] - ETA: 0s - loss: 1.7289 - CER: 0.1074 - WER: 0.2835
5426/5426 [=====] - 1058s 195ms/step - loss: 1.7289 - CER: 0.1074 - WER: 0.2835 - val_loss: 1.5612 - val_CER: 0.0939 - val_WER: 0.2463 - lr: 5.0000e-04
Epoch 65/1000
5426/5426 [=====] - ETA: 0s - loss: 1.7110 - CER: 0.1077 - WER: 0.2839
5426/5426 [=====] - 941s 173ms/step - loss: 1.7110 - CER: 0.1077 - WER: 0.2839 - val_loss: 1.3090 - val_CER: 0.0825 - val_WER: 0.2195 - lr: 5.0000e-04
Epoch 66/1000
5426/5426 [=====] - ETA: 0s - loss: 1.6869 - CER: 0.1055 - WER: 0.2812
5426/5426 [=====] - 885s 163ms/step - loss: 1.6869 - CER: 0.1055 - WER: 0.2812 - val_loss: 1.3896 - val_CER: 0.0858 - val_WER: 0.2263 - lr: 5.0000e-04
Epoch 67/1000
5426/5426 [=====] - ETA: 0s - loss: 1.6926 - CER: 0.1068 - WER: 0.2843
5426/5426 [=====] - 997s 184ms/step - loss: 1.6926 - CER: 0.1068 - WER: 0.2843 - val_loss: 1.3128 - val_CER: 0.0822 - val_WER: 0.2181 - lr: 5.0000e-04
Epoch 68/1000
867/5426 [====>.....] - ETA: 11:56 - loss: 1.6136 - CER: 0.1034 - WER: 0.2771
Package tensorflow installed (yesterday 143)
475/99 CRLF UTF-8 4spaces Python 3.11 (testRecognize)

```

Рис. 2.2. Процес тренування моделі

Проаналізуємо як навчилася модель, відкривши TensorBoard і переглянувши на метрики оцінки Character Error Rate (рис. 2.3) і Word Error Rate (рис. 2.4).

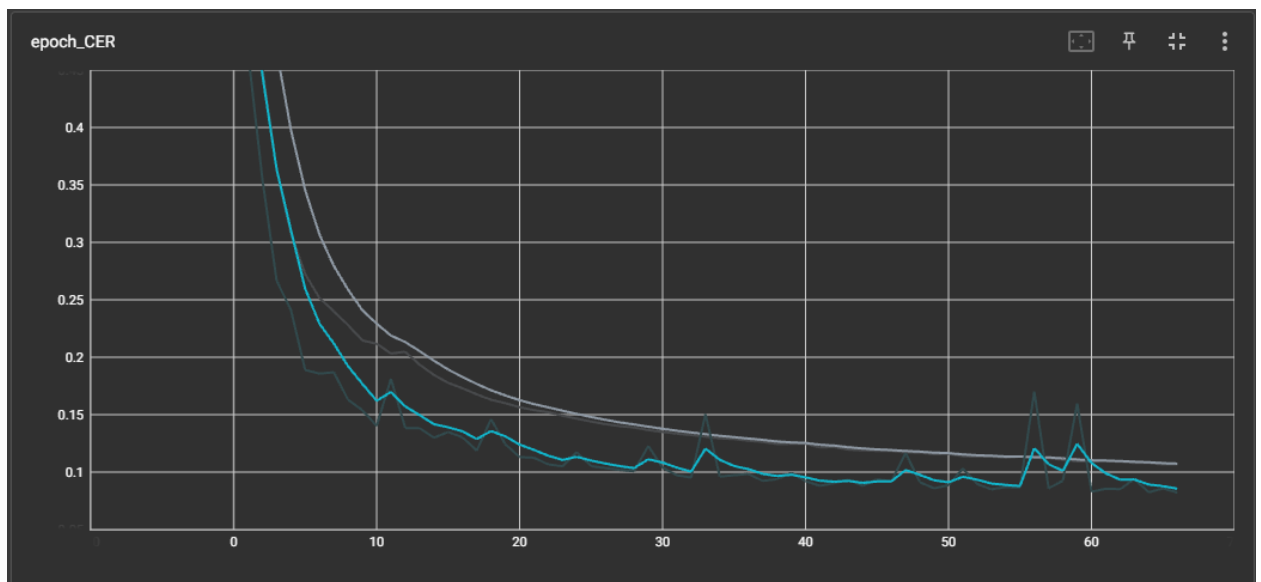


Рис. 2.3. Character Error Rate

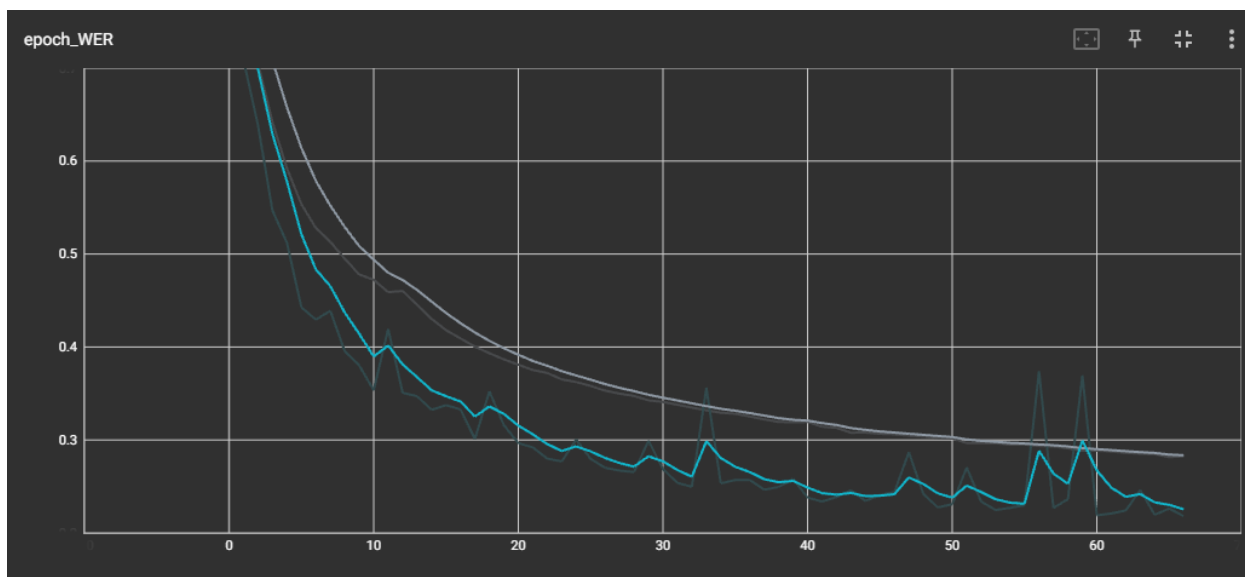


Рис. 2.4. Word Error Rate

З графіків бачимо поступове покращення передбачень моделі, яке в результаті навчання впродовж 67 епох досягла помилки в символах 0.1104 що означає що лише у 11.04% випадків символ передбачено неправильно. Цей результат є доволі високим. Помилки ж на рівні слів близько 30%. Крім цього можна зауважити аномальні стрибки значень графіку, причиною яких скоріше за все є присутність розділових знаків у наборі даних.

Після завершення процесу навчання проведемо тестування моделі на валідаційному наборі даних, щоб оцінити її продуктивність на даних, яких вона не бачила під час навчання. Для цього використаємо ті ж метрики які показує відсоток символів та слів, які модель прогнозує неправильно. Показник CER був обраний, оскільки він найбільш точно відображає точність розпізнавання на рівні символів. Програмний код для перевірки моделі `validation.py` наведено у додатку А. Процес перевірки на валідаційних даних з IAM Sentences та на зображеннях тексту на рисунках 3.1 та 3.2 відповідно.

```
validation x
Image: Datasets\IAM_Sentences\dataset\002-021-s02-00.png
Label: Only a few hours af-
Prediction: Only a few burs af .
CER: 0.2; WER: 0.6
100%|██████████| 1406/1409 [01:59<00:00, 11.52it/s]Image: Datasets\IAM_Sentences\dataset\006-119-s00-02.png
Label: time by more than 15,000,000 # last
Prediction: time by more than 18, 000,000 H last
CER: 0.08571428571428572; WER: 0.42857142857142855
100%|██████████| 1408/1409 [01:59<00:00, 11.26it/s]Image: Datasets\IAM_Sentences\dataset\b04-134-s02-00.png
Label: Thousands march with him - and sit with him
Prediction: Hhousands warch wth him - and it with hien
CER: 0.13953488372093023; WER: 0.5555555555555556
100%|██████████| 1409/1409 [01:59<00:00, 11.77it/s]
Image: Datasets\IAM_Sentences\dataset\001-122u-s02-00.png
Label: The health
Prediction: The health
CER: 0.0; WER: 0.0
Average CER: 0.11043651752908801, Average WER: 0.3273870006659219

Process finished with exit code 0
```

Рис. 2.5. Визначення точності моделі

```
Vesuvius seems to be tired ; he
1%|          | 16/1409 [01:05<42:33, 1.83s/it]Image: Datasets\IAM_Sentences\dataset\g06-011n-s03-00.png
Label: Vesuvius seems to be tired ; he
Prediction: Vesuvius seems to be tired ; he
CER: 0.0; WER: 0.0
```

Рис. 2.6. Приклад розпізнавання тексту зображення

Цей результат є досить хорошим для задачі розпізнавання рукописного тексту, враховуючи складність розпізнавання різних почерків та можливих варіацій у написанні символів. Проте модель має потенціал для подальшого покращення шляхом оптимізації даних та налаштувань параметрів.

Розширення набору даних: навчання моделі на більшому або більш різноманітному наборі даних може покращити її здатність до узагальнення. Додавання додаткових шарів або параметрів а також використання інших оптимізаторів може покращити продуктивність моделі. Використання методів доповнення, таких як додавання шуму або перетворення зображень, допоможе моделі краще узагальнювати дані. Крім цього в майбутньому модель можна модифікувати для розпізнавання рукописів з інших мов.

В подальшому буде розроблено функціонал API яке прийматиме фотографію рукописного тексту і надсилатиме у відповідь розпізнаний текст

та сам мобільний застосунок який взаємодіятиме з серверною частиною у вигляді API.

Висновки до другого розділу

У другому розділі було детально розглянуто процес підготовки набору даних IAM Sentences для навчання моделі розпізнавання рукописного тексту, включаючи читання та розбір даних, побудову та навчання моделі на основі згорткових та рекурентних нейронних мереж. Навчання моделі проводилося з використанням функції втрат CTC та оптимізатора Adam та дозволило досягти високої точності розпізнавання. Оцінка якості моделі проводилася за допомогою метрики CER та інструментарію TensorBoard та показала ефективність розробленої системи.

Було проаналізовано результати роботи моделі розпізнавання рукописного тексту, в результаті тестування на валідаційному наборі даних. Було встановлено, що середній CER у 11.04% свідчить про високу точність моделі, проте має потенціал для подальшого покращення. Розглянуто можливі напрямки для подальшого розвитку та оптимізації системи, що допоможе покращити її продуктивність та розширити спектр застосувань

Отримавши навчену модель можемо перейти до безпосередньої розробки мобільного застосунку, його клієнтської та серверної частини.

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗАСТОСУВАННЯ АЛГОРИТМУ

3.1. Розробка мобільного застосунку

Додаток складається з мобільного Android-застосунку та серверної частини, що виконує обробку зображень. Логіка роботи описана далі.

Обробка результату зйомки:

```
private val contract =
registerForActivityResult(ActivityResultContracts.TakePicture()) {
    photoImageView.setImageURI(null)
    photoImageView.setImageURI(imageUri)
    // Встановлюємо стандартний текст, поки не отримано нову відповідь
    recognizedTextView.text = getString(R.string.prediction_text)
    sendImageToApi(imageUri)
}
```

Першим кроком очищається попереднє зображення у photoImageView. Після отримання створеного шляху встановлюється нове зображення. Поле recognizedTextView оновлюється стандартним текстом в очікуванні результату розпізнавання з серверу. Викликається метод sendImageToApi, щоб передати зображення на сервер.

В методі onCreate [34] ініціалізуються UI-елементи та змінні, запитується дозвіл на використання камери, налаштовується обробка натискання на takePhotoButton для запуску камери та recognizedTextView для копіювання тексту в буфер обміну.

```
@RequiresApi(Build.VERSION_CODES.M)
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    takePhotoButton = findViewById(R.id.takePhotoButton)
    photoImageView = findViewById(R.id.photoImageView)
    recognizedTextView = findViewById(R.id.recognizedTextView)
    imageUri = createImageUri()

    // Запит дозволу на камеру
    ActivityCompat.requestPermissions(this,
        arrayOf(Manifest.permission.CAMERA), 1)

    takePhotoButton.setOnClickListener {
        contract.launch(imageUri)
    }

    // Додаємо функціонал для копіювання тексту
    recognizedTextView.setOnClickListener {
```

```

        val textToCopy = recognizedTextView.text.toString()
        if (textToCopy.isNotEmpty()) {
            val clipboard = getSystemService(Context.CLIPBOARD_SERVICE) as
ClipboardManager
            val clip = ClipData.newPlainText("Copied Text", textToCopy)
            clipboard.setPrimaryClip(clip)
            Toast.makeText(this, "Text copied", Toast.LENGTH_SHORT).show()
        } else {
            Toast.makeText(this, "No text to copy",
Toast.LENGTH_SHORT).show()
        }
    }
}

```

В методі `createImageUri` створюється файл для збереження фотографії у внутрішній пам'яті програми та повертається URI, який використовується камерою для запису зображення.

```

private fun createImageUri(): Uri {
    val image = File(filesDir, "photos.jpeg")
    return FileProvider.getUriForFile(
        this,
        "com.mrorny.cameraviewapp.FileProvider",
        image
    )
}

```

Зображення перед відправленням масштабуються (1280 пікселів у ширину) для оптимізації завантаження.

```

private fun resizeImage(originalBitmap: Bitmap, targetWidth: Int): Bitmap {
    // Розрахунок нової висоти зберігаючи пропорції
    val aspectRatio = originalBitmap.height.toFloat() /
originalBitmap.width.toFloat()
    val targetHeight = (targetWidth * aspectRatio).toInt()

    // Масштабування зображення до нових розмірів
    return Bitmap.createScaledBitmap(originalBitmap, targetWidth,
targetHeight, true)
}

```

Застосунок зчитує зображення з URI, масштабує його та перетворює у байтовий формат. Після цього формує HTTP-запит до API (POST) із зображенням у форматі Multipart [35] та обробляє відповідь сервера – У разі успіху виводить розпізнаний текст, у разі помилки відображає код помилки або повідомлення про збій.

```

private fun sendImageToApi(fileUri: Uri) {
    val client = OkHttpClient()
    val apiUrl = "https://htrapi.onrender.com/process-image/"

    try {
        // Відкриваємо файл з URI та створюємо Bitmap
        val inputStream = contentResolver.openInputStream(fileUri)
        val originalBitmap = BitmapFactory.decodeStream(inputStream)

        // Масштабуємо зображення до ширини 1280
        val resizedBitmap = resizeImage(originalBitmap, 1280)
    }
}

```

```

        // Перетворюємо Bitmap в байти
        val byteArrayOutputStream = ByteArrayOutputStream()
        resizedBitmap.compress(Bitmap.CompressFormat.JPEG, 85,
        byteArrayOutputStream)
        val fileBytes = byteArrayOutputStream.toByteArray()

        val requestBody = MultipartBody.Builder()
            .setType(MultipartBody.FORM)
            .addFormDataPart(
                "file", // Назва поля у запиті, як очікує сервер
                "photo.jpeg", // Ім'я файлу
                RequestBody.create(
                    "image/jpeg".toMediaType(),
                    fileBytes
                ) // Тіло файлу з MIME-типом
            )
            .build()

        // Створюємо HTTP-запит
        val request = Request.Builder()
            .url(apiUrl)
            .post(requestBody)
            .build()

        // Виконуємо запит в окремому потоці
        Thread {
            try {
                val response = client.newCall(request).execute()
                val responseBody = response.body?.string()

                if (response.isSuccessful && responseBody != null) {
                    // Парсимо відповідь JSON
                    val predictionText = parsePrediction(responseBody)

                    runOnUiThread {
                        recognizedTextView.text = predictionText
                    }
                } else {
                    runOnUiThread {
                        recognizedTextView.text =
                            "Error: ${response.code} - ${response.message}"
                    }
                }
            } catch (e: Exception) {
                runOnUiThread {
                    recognizedTextView.text = "Request failed: ${e.message}"
                }
            }
        }.start()
    } catch (e: Exception) {
        recognizedTextView.text = "Failed to send image: ${e.message}"
    }
}

```

3.2. Розробка серверної частини

Сервер, розгорнутий на Render [36], виконує ключові операції для обробки зображення та розпізнавання тексту. API побудований на основі FastAPI. Використовуються CORS-заголовки [37] для доступу з мобільного застосунку.

Визначаємо адреси джерел, які матимуть доступ до API:

```
app.add_middleware(
    CORSMiddleware,
    allow_origins=['*'],
    allow_credentials=True,
    allow_methods=["GET", "POST", "PUT", "DELETE"],
    allow_headers=["*"],
)
```

Створимо клас, відповідальний за передбачення тексту:

```
class ImageToWordModel(BaseONNXModel):
    def __init__(self, char_list: typing.Union[str, list], *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.char_list = char_list

    def predict(self, image: np.ndarray):
        image = resize_with_aspect_ratio(image,
            *self.input_shapes[0][1:3][::-1])
        image_pred = np.expand_dims(image, axis=0).astype(np.float32)
        preds = self.model.run(self.output_names, {self.input_names[0]:
            image_pred})[0]
        text = ctc_decoder(preds, self.char_list)[0]

        return text
```

Основна точка доступу: `/process-image/`. Опишемо логіку роботи API. Приймається зображення у форматі `image/jpeg`; Перевіряється MIME-тип [38] файлу, щоб упевнитись, що надсилається саме зображення. Після цього зображення декодується з байтового формату в масив NumPy, виконується перетворення у відтінки сірого та бінаризація для виділення тексту, застосовується морфологічна операція (dilation) [39], щоб об'єднати розділені елементи тексту, вирізається область із текстом, використовуючи контури. Підготовлене зображення масштабується до вхідного розміру нейронної мережі і викликається метод передбачення. Нейронна мережа обробляє зображення та повертає прогнозовані символи (індекси ймовірностей).

```
@app.post("/process-image/")
async def process_image(file: UploadFile = File(...)):
    # Логування типу контенту
    if not file.content_type:
        raise HTTPException(status_code=400, detail="Uploaded file does not
            have a content type.")

    if not file.content_type.startswith("image/"):
        raise HTTPException(status_code=400, detail=f"Uploaded file is not an
            image. Content type: {file.content_type}")

    try:
        # Зчитуємо вміст зображення
        image_content = await file.read()
        np_arr = np.frombuffer(image_content, np.uint8)
        image = cv2.imdecode(np_arr, cv2.IMREAD_COLOR)
```

```

        if image is None:
            raise HTTPException(status_code=400, detail="Failed to decode
image using OpenCV.")

        # Перетворення у відтінки сірого
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        # Бінаризація
        _, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)

        # Морфологічна операція для з'єднання близьких елементів тексту
        kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))
        dilated = cv2.dilate(thresh, kernel, iterations=2)

        # Знаходження контурів тексту та обрізання зайвого
        contours, _ = cv2.findContours(dilated, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
        all_contours = np.vstack([contour for contour in contours])
        x, y, w, h = cv2.boundingRect(all_contours)
        cropped = image[y:y + h, x:x + w]

```

Для декодування використовується функція `ctc_decoder`, що видаляє повтори символів і порожні мітки. В результаті повертається JSON [40] файл із розпізнаним текстом. В разі помилки сервер повертає код помилки та детальний опис.

```

prediction_text = model.predict(cropped)

return {"Prediction": {prediction_text}}

except Exception as e:
    raise HTTPException(status_code=500, detail=f"An error occurred:
{str(e)}")

```

Клас `BaseONNXModel` (див. додаток Г) завантажує ONNX-модель за допомогою `onnxruntime` [41]. Він автоматично визначає доступний пристрій (CPU або GPU) для обчислень, забезпечує базову функцію `predict`, яка викликається у дочірніх класах.

Модуль `utils.py` реалізує функцію `ctc_decoder` – групує однакові символи та видаляє порожні мітки та `resize_with_aspect_ratio`, що забезпечує масштабування зображення до заданих розмірів із додаванням чорних полів, щоб уникнути викривлення (див. додаток Д).

3.3. Демонстрація процесу роботи з додатком

Користувач відкриває додаток, на екрані відображаються кнопка для зйомки, поле для тексту та область для зображення. За замовчуванням поле зображення містить стандартне зображення, щоб користувач інтуїтивно зрозумів структуру інтерфейсу.



Рис.3.1. Головний екран застосунку

При першому запуску застосунку запитає дозвіл на використання камери. Наступним кроком користувач натискає кнопку «Take Photo», яка в свою чергу відкривається камера пристрою. Використання стандартного застосунку для зйомки є доволі влучним, оскільки користувачеві не потрібно пристосовуватися до додаткових незнайомих інтерфейсів.

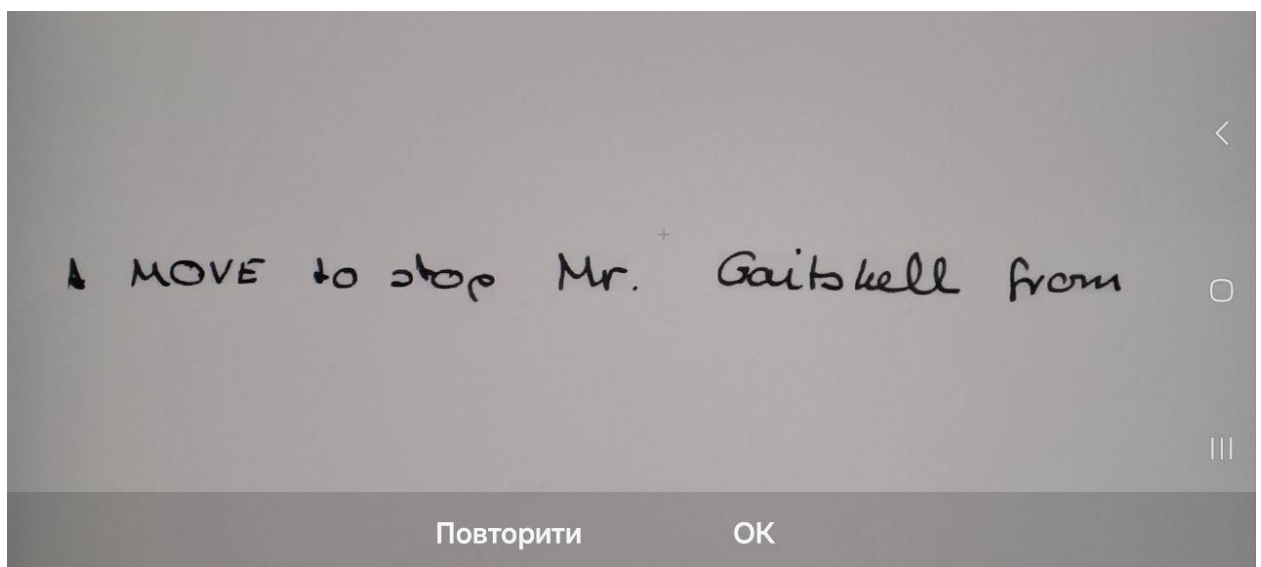


Рис.3.2. Процес зйомки

Після того, як користувач фотографує рукописний текст та підтверджує свій вибір вдалого кадру, зображення зберігається і відображається у ImageView на головному екрані. Після обробки сервером текст автоматично з'являється в полі TextView.

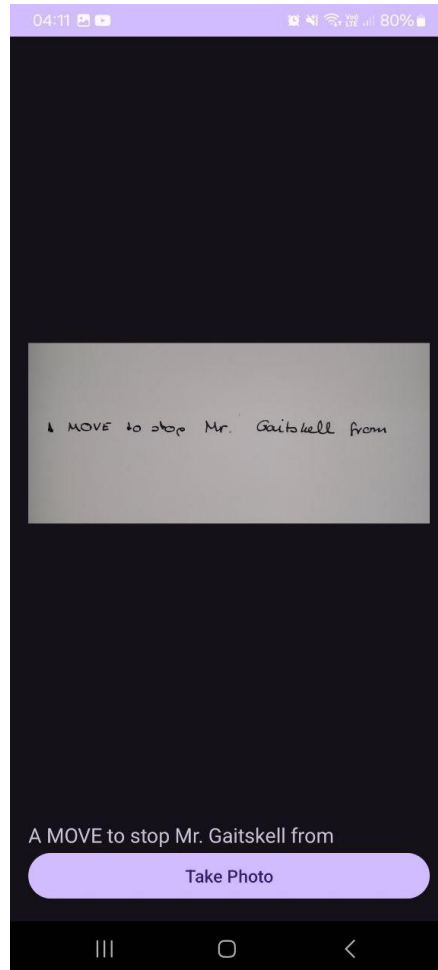


Рис.3.3. Розпізнаний текст

Для копіювання, користувач натискає на текст, щоб скопіювати його у буфер обміну, і отримує повідомлення "Text copied".

На даному етапі функціонал застосунку обмежується цими діями, проте головною метою була реалізація ефективної нейронної мережі, яка працюватиме в мобільному додатку, чого ми успішно досягли. Надалі доцільно розширити функціонал, додавши редагування та можливі інтеграції в застосунок для покращення користувацького досвіду.

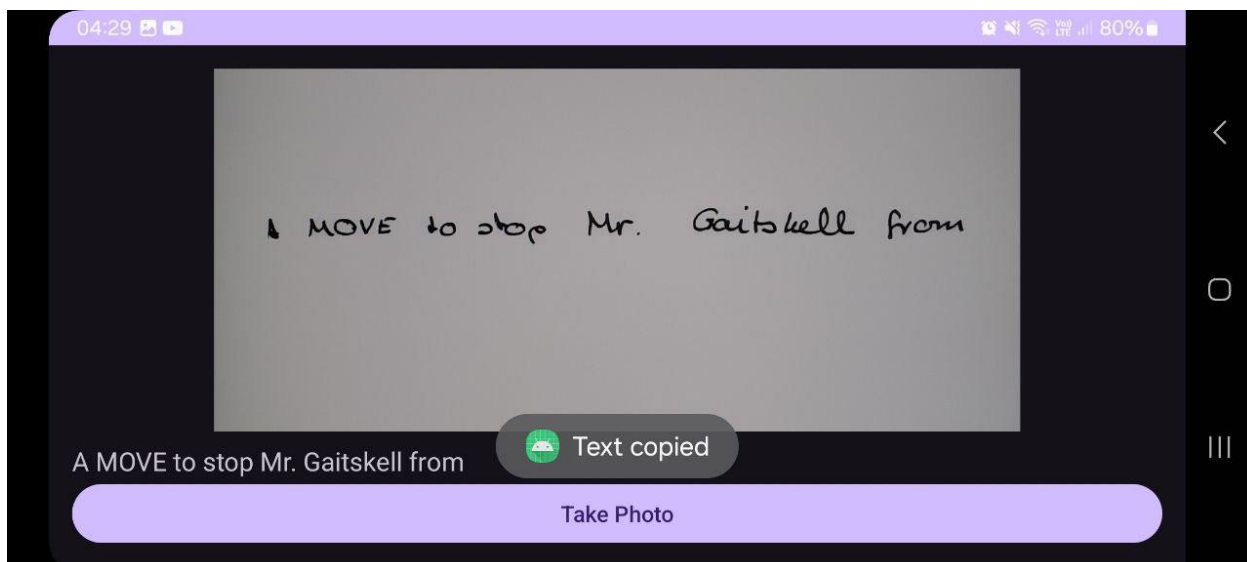


Рис.3.4. Копіювання тексту до буферу обміну

Висновки до третього розділу

У третьому розділі було детально описано процес розробки мобільного застосунку та серверної частини (API), а також представлено процес роботи продукту. Розробка мобільного додатку включала створення інтуїтивного інтерфейсу, налаштування взаємодії з камерою, реалізацію функції копіювання тексту та інтеграцію з API для обробки зображень. Було забезпечено коректну обробку користувацьких запитів, включаючи виведення розпізнаного тексту та обробку помилок.

Серверна частина розроблена на базі FastAPI та виконує всі необхідні операції для обробки зображень і розпізнавання тексту за допомогою нейронної мережі. Особливу увагу було приділено оптимізації обробки запитів, включаючи попередню обробку зображень (масштабування, бінаризація) та використання сучасних алгоритмів розпізнавання тексту (СТС).

Продемонстрована робота продукту показала, що застосунок успішно виконує поставлені задачі: здійснює фотографування рукописного тексту, надсилає зображення на сервер для обробки, відображає розпізнаний текст у зручному для користувача форматі.

ВИСНОВКИ

У рамках кваліфікаційної роботи магістра було виконано дослідження предметної області, спроектовано та реалізовано програмне забезпечення для розпізнавання рукописного тексту, для чого було вирішено усі поставлені завдання.

Було проведено аналіз предметної області та існуючих аналогів програм для оцифрування текстів. У результаті дослідження було розглянуто такі продукти, як Microsoft OneNote та Adobe Scan, їхній функціонал, переваги й недоліки. Це дозволило виявити сильні сторони та проблемні аспекти існуючих рішень, визначити ключові вимоги до розроблюваного програмного забезпечення та обґрунтувати вибір напрямків розробки.

Виконано дослідження сучасних методів та алгоритмів розпізнавання тексту, включаючи згорткові нейронні мережі (CNN), рекурентні нейронні мережі (RNN) та функцію Connectionist Temporal Classification (CTC). Обрані підходи забезпечують високу ефективність обробки зображень без попередньої сегментації символів, що є оптимальним для задачі розпізнавання рукописного тексту.

Виконано аналіз інструментальних засобів розробки. Для мобільної частини додатку обрано Kotlin та Android Studio, що дозволило створити стабільний, продуктивний та масштабований застосунок для платформи Android. Для серверної частини використано FastAPI, що забезпечило високу продуктивність, легкість розгортання та можливість інтеграції з моделлю нейронної мережі.

Реалізовано програмний продукт, який складається з мобільного додатку та серверної частини (API). У мобільному додатку створено інтуїтивний інтерфейс, реалізовано функції зйомки фотографії, надсилання зображень на сервер, отримання та відображення розпізнаного тексту. Серверна частина виконує попередню обробку зображень (масштабування,

бінаризацію) та розпізнавання тексту за допомогою попередньо навчених нейронних моделей.

Продемонстровано роботу розробленого продукту. В результаті тестування підтверджено, що додаток успішно виконує поставлені задачі: забезпечує розпізнавання тексту з високою точністю, дозволяє зберігати конфіденційність даних та спрощує процес оцифрування рукописних матеріалів.

Розроблений продукт повністю відповідає поставленим вимогам і реалізує всі необхідні функції. Його результативність підтверджує ефективність використаних підходів і технологій. Є можливість вдосконалення застосунку в майбутньому, зокрема додаванням підтримки більшої кількості мов, детальнішою обробкою зображення перед застосування моделі передбачення та забезпеченням більшого функціоналу подальшої обробки отриманого розпізнаного тексту.

Протягом виконання роботи було отримано нові знання та навички з розробки програмного забезпечення, машинного навчання, обробки текстових даних, проєктування мобільних додатків та API. Досвід роботи з нейронними мережами, вибір оптимальних архітектур та інтеграція алгоритмів машинного навчання у програмні рішення показали високу потенційність подальшого застосування таких методів.

Результати роботи свідчать про перспективність розробки подібних продуктів для автоматизації роботи з рукописними текстами, що може знайти застосування в освіті, бізнесі, науці та інших сферах, де потрібне оцифрування текстової інформації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Леут Назарій., Вакалюк Тетяна. Мобільний додаток для розпізнавання рукописного тексту / Збірник матеріалів XI Всеукраїнської науково-практичної конференції молодих вчених «Наукова молодь-2023» (Київ, 21 листопада 2023 р.). / упоряд.: А. Яцишин. К.: ЦП «КОМПРИНТ», 2023. 338 с.
2. Leut N. THE USE OF RECURRENT NEURAL NETWORKS FOR HANDWRITTEN TEXT RECOGNITION / Збірник тез доповідей II Міжнар. наук.-практич. конфер. «Інновації та перспективні шляхи розвитку інформаційних технологій» (06 груд. 2023 р., м. Черкаси) [Електронний ресурс] / упоряд. : Т. О. Прокопенко, Я. В. Тарасенко ; М-во освіти і науки України, Черкас. держ. технол. ун-т. – Черкаси : ЧДТУ, 2023. – 277 с.
3. Neil T. Mobile Design Pattern Gallery: UI Patterns for Smartphone Apps. – O'Reilly Media, 2014. – 404 с.
4. Hellman E. Android Programming: Pushing the Limits. – Wiley & Sons, Incorporated, John, 2013. – 432 с.
5. OCR vs HTR: The Key Differences Between Optical Character Recognition and Handwritten Text Recognition [Електронний ресурс]. – Режим доступу до ресурсу: <https://readcoop.eu/insights/ocr-vs-htr/>
6. Microsoft OneNote [Електронний ресурс] // Microsoft. – 2024. – Режим доступу до ресурсу: <https://www.microsoft.com/uk-ua/microsoft-365/onenote/digital-note-taking-app>
7. Adobe Scan для мобільних пристроїв [Електронний ресурс] // Adobe. – 2024. – Режим доступу до ресурсу: <https://www.adobe.com/ua/acrobat/mobile/scanner-app.html>
8. How to easily do Handwriting Recognition using Machine Learning [Електронний ресурс] // Nanonets. – 2024. – Режим доступу до ресурсу: <https://nanonets.com/blog/handwritten-character-recognition/>

9. Template Matching [Электронный ресурс] // Adaptive Vision Studio Documentation. – Режим доступа до ресурсу: https://docs.adaptive-vision.com/4.7/studio/machine_vision_guide/TemplateMatching.html

10. Feature Extraction [Электронный ресурс] // MathWorks. – Режим доступа до ресурсу: <https://www.mathworks.com/discovery/feature-extraction.html>

11. Suganthan P.N. Feature Selection for Classification: A Review / P.N. Suganthan, L. Fausett // Artificial Intelligence Review. – 2010. – Vol. 34, No. 3. – С. 259–276. – Режим доступа до ресурсу: <https://link.springer.com/article/10.1007/s10462-010-9154-1>

12. k-Nearest Neighbors (k-NN) [Электронный ресурс] // IBM. – Режим доступа до ресурсу: <https://www.ibm.com/topics/knn>

13. Cortes C. Support-Vector Networks / C. Cortes, V. Vapnik // Machine Learning. – 1995. – Vol. 20, No. 3. – С. 273–297. – Режим доступа до ресурсу: http://image.diku.dk/imagecanon/material/cortes_vapnik95.pdf

14. Goodfellow I. Deep Learning [Электронный ресурс] / I. Goodfellow, Y. Bengio, A. Courville // MIT Press. – 2016. – Режим доступа до ресурсу: <https://www.deeplearningbook.org/>

15. LeCun Y. Deep Learning / Y. LeCun, Y. Bengio, G. Hinton // Nature. – 2015. – Vol. 521, No. 7553. – С. 436–444. – Режим доступа до ресурсу: <https://www.nature.com/articles/nature14539>

16. CS231n: Convolutional Neural Networks for Visual Recognition [Электронный ресурс]. – Stanford University. – Режим доступа до ресурсу: <https://cs231n.stanford.edu/>

17. Convolutional Neural Networks (CNN) [Электронный ресурс] // IBM. – Режим доступа до ресурсу: <https://www.ibm.com/topics/convolutional-neural-networks>

18. Introduction to Recurrent Neural Networks (RNN) [Электронный ресурс] // GeeksforGeeks. – Режим доступа до ресурсу: <https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/>

19. What is REST? [Электронный ресурс] // RESTful API. – 2024. – Режим доступа до ресурсу: <https://restfulapi.net/>
20. About Python [Электронный ресурс] // Python Software Foundation. – 2024. – Режим доступа до ресурсу: <https://www.python.org/about/>
21. pandas - Python Data Analysis Library [Электронный ресурс] // Pandas. – 2024. – Режим доступа до ресурсу: <https://pandas.pydata.org/>
22. About – OpenCV [Электронный ресурс] // OpenCV. – 2024. – Режим доступа до ресурсу: <https://opencv.org/about/>
23. Image processing in Python [Электронный ресурс] // Scikit-image. – 2024. – Режим доступа до ресурсу: <https://scikit-image.org/>
24. FastAPI Features [Электронный ресурс] // FastAPI. – 2024. – Режим доступа до ресурсу: <https://fastapi.tiangolo.com/features/>
25. Getting Started with Kotlin [Электронный ресурс] // Kotlin. – 2024. – Режим доступа до ресурсу: <https://kotlinlang.org/docs/getting-started.html#install-kotlin>
26. Android Studio [Электронный ресурс] // Android Developers. – 2024. – Режим доступа до ресурсу: <https://developer.android.com/studio>
27. IAM Handwriting Database [Электронный ресурс]. – Режим доступа до ресурсу: <https://fki.tic.heia-fr.ch/databases/iam-handwriting-database>.
28. Discover LSTM [Электронный ресурс] // NVIDIA Developer. – 2024. – Режим доступа до ресурсу: <https://developer.nvidia.com/discover/lstm>
29. The Role of Softmax in Neural Networks: Detailed Explanation and Applications [Электронный ресурс] // GeeksforGeeks. – 2024. – Режим доступа до ресурсу: <https://www.geeksforgeeks.org/the-role-of-softmax-in-neural-networks-detailed-explanation-and-applications/>
30. Adam Optimizer [Электронный ресурс] // Keras. – 2024. – Режим доступа до ресурсу: <https://keras.io/api/optimizers/adam/>
31. WER, CER, MER Metrics [Электронный ресурс] // Kolena. – 2024. – Режим доступа до ресурсу: <https://docs.kolena.com/metrics/wer-cer-mer/>

32. Keras Callbacks API [Электронный ресурс] // TensorFlow. – 2024. – Режим доступа до ресурсу: https://www.tensorflow.org/api_docs/python/tf/keras/callbacks
33. TensorBoard [Электронный ресурс] // TensorFlow. – 2024. – Режим доступа до ресурсу: <https://www.tensorflow.org/tensorboard>
34. Activity Lifecycle [Электронный ресурс] // Android Developers. – 2024. – Режим доступа до ресурсу: <https://developer.android.com/guide/components/activities/activity-lifecycle>
35. Multipart Form Data [Электронный ресурс] // Adam Chalmers Blog. – 2024. – Режим доступа до ресурсу: <https://blog.adamchalmers.com/multipart/>
36. About Render [Электронный ресурс] // Render. – 2024. – Режим доступа до ресурсу: <https://render.com/about>
37. Cross-Origin Resource Sharing (CORS) [Электронный ресурс] // MDN Web Docs. – 2024. – Режим доступа до ресурсу: <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>
38. MIME Types [Электронный ресурс] // MDN Web Docs. – 2024. – Режим доступа до ресурсу: https://developer.mozilla.org/en-US/docs/Web/HTTP/MIME_types
39. Erosion and Dilation Tutorial [Электронный ресурс] // OpenCV. – 2024. – Режим доступа до ресурсу: https://docs.opencv.org/3.4/db/df6/tutorial_erosion_dilatation.html
40. JSON Specification [Электронный ресурс] // JSON.org. – 2024. – Режим доступа до ресурсу: <https://www.json.org/json-en.html>
41. ONNX Runtime [Электронный ресурс] // ONNX Runtime. – 2024. – Режим доступа до ресурсу: <https://onnxruntime.ai/>

ДОДАТКИ

Програмний код тренування train.py

```

import os
import pandas as pd
import tensorflow as tf
from keras import layers
from keras.models import Model
from keras.callbacks import EarlyStopping, ModelCheckpoint,
ReduceLROnPlateau, TensorBoard
from keras.utils import pad_sequences
from sklearn.model_selection import train_test_split
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from tqdm import tqdm
import cv2
import numpy as np
from configs import ModelConfigs
from residual_block import residual_block

# Завантаження та обробка датасету
sentences_txt_path = os.path.join("Datasets", "IAM_Sentences", "metadata",
"sentences.txt")
sentences_folder_path = os.path.join("Datasets", "IAM_Sentences", "dataset")

# Змінні для набору даних
dataset, vocab, max_length = [], set(), 0
words = open(sentences_txt_path, "r").readlines()

for line in tqdm(words):
    if line.startswith("#"):
        continue

    line_split = line.split(" ")
    if line_split[2] == "err":
        continue

    file_name = line_split[0] + ".png"
    label = line_split[-1].rstrip("\n").replace("|", " ")

    rel_path = os.path.join(sentences_folder_path, file_name)
    if not os.path.exists(rel_path):
        print(f"Файл не знайдено: {rel_path}")
        continue

    dataset.append([rel_path, label])
    vocab.update(list(label))
    max_length = max(max_length, len(label))

# Створення об'єкту конфігурацій
configs = ModelConfigs()
configs.vocab = "".join(sorted(vocab))
configs.max_text_length = max_length
configs.save()

# Поділ на навчальний та валідаційний набори
data_df = pd.DataFrame(dataset, columns=["image_path", "label"])
train_df, val_df = train_test_split(data_df, test_size=0.2, random_state=42)

train_df.to_csv(os.path.join(configs.model_path, "train.csv"), index=False)
val_df.to_csv(os.path.join(configs.model_path, "val.csv"), index=False)

# Доповнення даних

```

```

train_data_gen = ImageDataGenerator(
    brightness_range=[0.8, 1.2],
    zoom_range=0.2,
    preprocessing_function=lambda x: cv2.GaussianBlur(x, (5, 5), 0)
)

# Створення словника індексації символів
char_to_num = {char: idx for idx, char in enumerate(configs.vocab)}

# Функція для перетворення тексту в числовий формат
def text_to_sequence(text):
    return [char_to_num[char] for char in text if char in char_to_num]

# Завантаження та підготовка зображень для тренування
train_images, train_labels = [], []

for index, row in train_df.iterrows():
    image_path, label = row["image_path"], row["label"]
    try:
        image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
        if image is None:
            print(f"Could not read image: {image_path}")
            continue

        image = cv2.resize(image, (configs.width, configs.height))
        image = image / 255.0 # Нормалізація

        train_images.append(np.expand_dims(image, axis=-1))
        train_labels.append(text_to_sequence(label))

    except Exception as e:
        print(f"Error processing image {image_path}: {e}")

train_images = np.array(train_images)
train_labels = np.array(train_labels)

# Завантаження та підготовка валідаційного набору
val_images, val_labels = [], []

for index, row in val_df.iterrows():
    image_path, label = row["image_path"], row["label"]
    try:
        image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
        if image is None:
            print(f"Could not read image: {image_path}")
            continue

        image = cv2.resize(image, (configs.width, configs.height))
        image = image / 255.0

        val_images.append(np.expand_dims(image, axis=-1))
        val_labels.append(text_to_sequence(label))

    except Exception as e:
        print(f"Error processing image {image_path}: {e}")

val_images = np.array(val_images)

# Перетворення текстових міток у числовий формат із падінгом
train_labels = [text_to_sequence(label) for label in train_df["label"]]
val_labels = [text_to_sequence(label) for label in val_df["label"]]

```

```

# Застосування паддингу
train_labels = pad_sequences(train_labels, maxlen=configs.max_text_length,
padding="post", value=-1)
val_labels = pad_sequences(val_labels, maxlen=configs.max_text_length,
padding="post", value=-1)

# Підготовка наборів даних
train_dataset = tf.data.Dataset.from_tensor_slices((train_images,
train_labels))
train_dataset =
train_dataset.batch(configs.batch_size).prefetch(tf.data.AUTOTUNE)

val_dataset = tf.data.Dataset.from_tensor_slices((val_images, val_labels))
val_dataset =
val_dataset.batch(configs.batch_size).prefetch(tf.data.AUTOTUNE)

# Функція CTC Loss
def ctc_loss(y_true, y_pred):
    y_pred = tf.transpose(y_pred, (1, 0, 2))
    label_length = tf.math.reduce_sum(tf.cast(y_true != -1, tf.int32), axis=-
1)
    input_length = tf.fill([tf.shape(y_pred)[1]], tf.shape(y_pred)[0])
    loss = tf.nn.ctc_loss(
        labels=tf.cast(y_true, tf.int32),
        logits=y_pred,
        label_length=label_length,
        logit_length=input_length,
        blank_index=len(configs.vocab)
    )
    return tf.reduce_mean(loss)

# Метрики CER і WER
class MetricsLogger(tf.keras.callbacks.Callback):
    def __init__(self, val_data, char_list):
        super().__init__()
        self.val_data = val_data
        self.char_list = char_list

    def on_epoch_end(self, epoch, logs=None):
        cer_scores, wer_scores = [], []
        for images, labels in self.val_data:
            predictions = self.model.predict(images)
            decoded_predictions = [
                ''.join([self.char_list[i] for i in tf.argmax(pred, axis=-
1).numpy() if i < len(self.char_list)])
                for pred in predictions
            ]
            for true_label, pred_label in zip(labels.numpy(),
decoded_predictions):
                true_text = ''.join([self.char_list[i] for i in true_label if
i != -1])
                cer_scores.append(self.character_error_rate(pred_label,
true_text))
                wer_scores.append(self.word_error_rate(pred_label, true_text))
        avg_cer = np.mean(cer_scores)
        avg_wer = np.mean(wer_scores)
        logs['val_CER'] = avg_cer
        logs['val_WER'] = avg_wer
        print(f"Epoch {epoch + 1} - val_CER: {avg_cer:.4f}, val_WER:
{avg_wer:.4f}")

    @staticmethod

```

```

def character_error_rate(pred, true):
    return MetricsLogger.levenshtein_distance(pred, true) / max(len(true),
1)

    @staticmethod
    def word_error_rate(pred, true):
        return MetricsLogger.character_error_rate(pred.split(), true.split())

    @staticmethod
    def levenshtein_distance(a, b):
        dp = np.zeros((len(a) + 1, len(b) + 1), dtype=np.int32)
        for i in range(len(a) + 1):
            dp[i][0] = i
        for j in range(len(b) + 1):
            dp[0][j] = j
        for i in range(1, len(a) + 1):
            for j in range(1, len(b) + 1):
                dp[i][j] = min(
                    dp[i - 1][j] + 1,
                    dp[i][j - 1] + 1,
                    dp[i - 1][j - 1] + (0 if a[i - 1] == b[j - 1] else 1)
                )
        return dp[len(a)][len(b)]

# Створення моделі
def create_model():
    inputs = layers.Input(shape=(configs.height, configs.width, 1),
name="input")
    x = layers.Lambda(lambda x: x / 255)(inputs)

    x = residual_block(x, 32, strides=1)
    x = residual_block(x, 32, strides=2)
    x = residual_block(x, 64, strides=2)
    x = residual_block(x, 128, strides=2)
    x = layers.Reshape((-1, x.shape[-1]))(x)

    x = layers.Bidirectional(layers.LSTM(128, return_sequences=True))(x)
    x = layers.Dropout(0.2)(x)
    x = layers.Bidirectional(layers.LSTM(64, return_sequences=True))(x)
    x = layers.Dropout(0.2)(x)

    outputs = layers.Dense(len(configs.vocab) + 1, activation="softmax",
name="output")(x)

    model = Model(inputs, outputs)
    model.compile(optimizer=Adam(learning_rate=configs.learning_rate),
loss=ctc_loss)
    return model

model = create_model()
model.summary()

# Зворотні виклики
callbacks = [
    EarlyStopping(patience=10, restore_best_weights=True),
    ModelCheckpoint(os.path.join(configs.model_path, "best_model.h5"),
save_best_only=True),
    ReduceLROnPlateau(patience=5, factor=0.5, verbose=1),
    TensorBoard(log_dir=os.path.join(configs.model_path, "logs")),
    MetricsLogger(val_data=val_dataset, char_list=configs.vocab)
]

```

```
# Тренування
model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=configs.train_epochs,
    callbacks=callbacks
)
```

Програмний код файлу валідації validation.py

```
import os
import cv2
import numpy as np
import pandas as pd
from keras.models import load_model
from configs import ModelConfigs
from tqdm import tqdm

from train import text_to_sequence
from utils import ctc_decoder, get_cer, get_wer

# Клас для роботи з моделлю розпізнавання тексту
class TextRecognitionModel:
    def __init__(self, model_path, char_list):
        self.model = load_model(model_path, compile=False)
        self.char_list = char_list

    def predict(self, image):
        image = cv2.resize(image, (configs.width, configs.height)) # Зміна
        # розміру зображення
        image = image / 255.0
        image = np.expand_dims(image, axis=(0, -1))
        predictions = self.model.predict(image)
        text = ctc_decoder(predictions, self.char_list)[0]
        return text

if __name__ == "__main__":
    # Завантаження конфігурацій
    configs = ModelConfigs.load("Models/HTRAPP")

    # Створення словника індексації символів
    char_to_num = {char: idx for idx, char in enumerate(configs.vocab)}
    num_to_char = {idx: char for char, idx in char_to_num.items()}

    # Завантаження моделі
    model = TextRecognitionModel(
        model_path=os.path.join(configs.model_path, "best_model.h5"),
        char_list=configs.vocab
    )

    # Завантаження валідаційного набору
    val_data = pd.read_csv(os.path.join(configs.model_path, "val.csv"))

    cer_scores, wer_scores = [], []

    # Оцінка моделі на валідаційному наборі
    for index, row in tqdm(val_data.iterrows(), total=len(val_data)):
        image_path, true_label = row["image_path"], row["label"]
        try:
            image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

            if image is None:
                print(f"Could not read image: {image_path}")
                continue

            true_sequence = text_to_sequence(true_label, char_to_num)
            predicted_text = model.predict(image)
```

```
cer = get_cer(predicted_text, true_label)
wer = get_wer(predicted_text, true_label)

print(f"Image: {image_path}")
print(f"True Label: {true_label}")
print(f"Predicted Text: {predicted_text}")
print(f"CER: {cer:.2f}, WER: {wer:.2f}\n")

cer_scores.append(cer)
wer_scores.append(wer)

except Exception as e:
    print(f"Error processing image {image_path}: {e}")

print(f"Average CER: {np.mean(cer_scores):.2f}, Average WER:
{np.mean(wer_scores):.2f}")
```

Програмний код файлу utils.py

```
import numpy as np

def ctc_decoder(predictions, char_list):
    """
    Функція для декодування CTC
    """
    decoded = []
    for pred in predictions:
        output_text = ""
        last_char = None
        for p in np.argmax(pred, axis=1):
            if p != last_char and p < len(char_list):
                output_text += char_list[p]
            last_char = p
        decoded.append(output_text)
    return decoded

def get_cer(pred_text, true_text):
    """
    Функція для оцінки CER (Character Error Rate)
    """
    dist = np.zeros((len(true_text) + 1, len(pred_text) + 1))
    for i in range(len(true_text) + 1):
        dist[i][0] = i
    for j in range(len(pred_text) + 1):
        dist[0][j] = j
    for i in range(1, len(true_text) + 1):
        for j in range(1, len(pred_text) + 1):
            cost = 0 if true_text[i - 1] == pred_text[j - 1] else 1
            dist[i][j] = min(dist[i - 1][j] + 1, dist[i][j - 1] + 1, dist[i - 1][j - 1] + cost)
    return dist[-1][-1] / len(true_text)

def get_wer(pred_text, true_text):
    """
    Функція для оцінки WER
    """
    pred_words = pred_text.split()
    true_words = true_text.split()
    return get_cer(pred_words, true_words)
```

Програмний код файлу API main.py

```

import uvicorn
from fastapi import FastAPI, File, UploadFile, HTTPException
from fastapi.middleware.cors import CORSMiddleware
import os
import cv2
import typing
import numpy as np
from BaseONNXModel import BaseONNXModel, BaseModelConfigs
from utils import ctc_decoder
from utils import resize_with_aspect_ratio

app = FastAPI()

# Визначаємо адреси джерел, які матимуть доступ до API
app.add_middleware(
    CORSMiddleware,
    allow_origins=['*'],
    allow_credentials=True,
    allow_methods=["GET", "POST", "PUT", "DELETE"],
    allow_headers=["*"],
)

# Клас реалізації передбачення
class ImageToWordModel(BaseONNXModel):
    def __init__(self, char_list: typing.Union[str, list], *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.char_list = char_list

    def predict(self, image: np.ndarray):
        image = resize_with_aspect_ratio(image, *self.input_shapes[0][1:3][::-1])
        image_pred = np.expand_dims(image, axis=0).astype(np.float32)
        preds = self.model.run(self.output_names, {self.input_names[0]:
image_pred})[0]
        text = ctc_decoder(preds, self.char_list)[0]

        return text

# Зчитуємо конфігурацію та завантажуюмо відповідну модель
configs =
BaseModelConfigs.load("Models/handwriting_recognizer/202301131202/configs.yam
l")
model = ImageToWordModel(model_path=configs.model_path,
char_list=configs.vocab)

@app.get("/")
async def root():
    return {"message": "Welcome to the HTRAPI"}

@app.post("/process-image/")
async def process_image(file: UploadFile = File(...)):
    # Логування типу контенту
    if not file.content_type:
        raise HTTPException(status_code=400, detail="Uploaded file does not
have a content type.")

```

```

    if not file.content_type.startswith("image/"):
        raise HTTPException(status_code=400, detail=f"Uploaded file is not an
image. Content type: {file.content_type}")

    try:
        # Зчитуємо вміст зображення
        image_content = await file.read()
        np_arr = np.frombuffer(image_content, np.uint8)
        image = cv2.imdecode(np_arr, cv2.IMREAD_COLOR)

        if image is None:
            raise HTTPException(status_code=400, detail="Failed to decode image
using OpenCV.")

        # Перетворення у відтінки сірого
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        # Бінаризація
        _, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)

        # Морфологічна операція для з'єднання близьких елементів тексту
        kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))
        dilated = cv2.dilate(thresh, kernel, iterations=2)

        # Знаходження контурів тексту та обрізання зайвого
        contours, _ = cv2.findContours(dilated, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
        all_contours = np.vstack([contour for contour in contours])
        x, y, w, h = cv2.boundingRect(all_contours)
        cropped = image[y:y + h, x:x + w]
        # Передбачення
        prediction_text = model.predict(cropped)

        return {"Prediction": {prediction_text}}

    except Exception as e:
        raise HTTPException(status_code=500, detail=f"An error occurred:
{str(e)}")

```

Програмний код файлу utils.py серверної частини

```

import typing
import numpy as np
from itertools import groupby
import cv2
from typing import Tuple

def ctc_decoder(predictions: np.ndarray, chars: typing.Union[str, list]) ->
typing.List[str]:
    """
    Жадібне декодування з використанням CTC
    """
    # вибір символів з максимальною ймовірністю
    argmax_preds = np.argmax(predictions, axis=-1)

    # групування однакових символів
    grouped_preds = [[k for k, _ in groupby(preds)] for preds in argmax_preds]

    # перетворення індексів в символи
    texts = ["".join([chars[k] for k in group if k < len(chars)]) for group in
grouped_preds]

    return texts

def resize_with_aspect_ratio(
    image: np.ndarray,
    width_target: int,
    height_target: int,
    padding_color: Tuple[int, int, int] = (0, 0, 0)
) -> np.ndarray:
    """
    Зміна розміру зображення зі збереженням співвідношення сторін
    """
    # оригінальні розміри
    height, width = image.shape[:2]

    # визначене співвідношення сторін
    ratio = min(width_target / width, height_target / height)
    new_width = int(width * ratio)
    new_height = int(height * ratio)

    # зміна розміру зображення
    resized_image = cv2.resize(image, (new_width, new_height))

    # розрахунок відступів
    delta_width = width_target - new_width
    delta_height = height_target - new_height
    top = delta_height // 2
    bottom = delta_height - top
    left = delta_width // 2
    right = delta_width - left

    # додавання відступів до зміненого зображення
    padded_image = cv2.copyMakeBorder(
        resized_image,
        top,
        bottom,
        left,
        right,

```

```
        cv2.BORDER_CONSTANT,  
        value=padding_color  
    )  
  
    return padded_image
```

Програмний код файлу activity_main.xml, відповідального за зовнішній
вигляд мобільного застосунку

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <ImageView
        android:id="@+id/photoImageView"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:scaleType="fitCenter"
        android:adjustViewBounds="true"
        android:contentDescription="@string/photo_placeholder" />

    <TextView
        android:id="@+id/recognizedTextView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/prediction_text"
        android:textSize="16sp"
        android:paddingTop="8dp" />

    <Button
        android:id="@+id/takePhotoButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/photo_button_text" />
</LinearLayout>
```

Програмний код файлу MainActivity.kt, відповідального за логіку роботи мобільного застосунку

```
package com.mrorny.cameraviewapp

import android.Manifest
import android.content.ClipData
import android.content.ClipboardManager
import android.content.Context
import android.graphics.Bitmap
import android.graphics.BitmapFactory
import android.net.Uri
import android.os.Build
import android.os.Bundle
import android.widget.Button
import android.widget.ImageView
import android.widget.TextView
import android.widget.Toast
import androidx.activity.result.contract.ActivityResultContracts
import androidx.annotation.RequiresApi
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.ActivityCompat
import androidx.core.content.FileProvider
import okhttp3.MediaType.Companion.toMediaType
import okhttp3.MultipartBody
import okhttp3.OkHttpClient
import okhttp3.Request
import okhttp3.RequestBody
import org.json.JSONException
import org.json.JSONObject
import java.io.ByteArrayOutputStream
import java.io.File

class MainActivity : AppCompatActivity() {

    private lateinit var takePhotoButton: Button
    private lateinit var photoImageView: ImageView
    private lateinit var recognizedTextView: TextView
    private val REQUEST_IMAGE_CAPTURE = 1
    private lateinit var imageUri: Uri
    private var photoBitmap: Bitmap? = null

    private val contract =
registerForActivityResult(ActivityResultContracts.TakePicture()) {
    photoImageView.setImageURI(null)
    photoImageView.setImageURI(imageUri)
    // Встановлюємо стандартний текст, поки не отримано нову відповідь
    recognizedTextView.text = getString(R.string.prediction_text)
    sendImageToApi(imageUri)
}

@RequiresApi(Build.VERSION_CODES.M)
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    takePhotoButton = findViewById(R.id.takePhotoButton)
    photoImageView = findViewById(R.id.photoImageView)
    recognizedTextView = findViewById(R.id.recognizedTextView)
    imageUri = createImageUri()
```

```

        // Запит дозволу на камеру
        ActivityCompat.requestPermissions(this,
            arrayOf(Manifest.permission.CAMERA), 1)

        takePhotoButton.setOnClickListener {
            contract.launch(imageUri)
        }

        // Додаємо функціонал для копіювання тексту
        recognizedTextView.setOnClickListener {
            val textToCopy = recognizedTextView.text.toString()
            if (textToCopy.isNotEmpty()) {
                val clipboard = getSystemService(Context.CLIPBOARD_SERVICE) as
ClipboardManager
                val clip = ClipData.newPlainText("Copied Text", textToCopy)
                clipboard.setPrimaryClip(clip)
                Toast.makeText(this, "Text copied", Toast.LENGTH_SHORT).show()
            } else {
                Toast.makeText(this, "No text to copy",
                    Toast.LENGTH_SHORT).show()
            }
        }
    }

    private fun createImageUri(): Uri {
        val image = File(filesDir, "photos.jpeg")
        return FileProvider.getUriForFile(
            this,
            "com.mrorny.cameraviewapp.FileProvider",
            image
        )
    }

    private fun resizeImage(originalBitmap: Bitmap, targetWidth: Int): Bitmap
    {
        // Розрахунок нової висоти зберігаючи пропорції
        val aspectRatio = originalBitmap.height.toFloat() /
originalBitmap.width.toFloat()
        val targetHeight = (targetWidth * aspectRatio).toInt()

        // Масштабування зображення до нових розмірів
        return Bitmap.createScaledBitmap(originalBitmap, targetWidth,
            targetHeight, true)
    }

    // Метод для парсингу тексту передбачення з JSON
    private fun parsePrediction(responseBody: String): String {
        return try {
            val jsonObject = JSONObject(responseBody)
            // Отримуємо значення ключа "Prediction" і перетворюємо на рядок
            jsonObject.getJSONArray("Prediction").getString(0)
        } catch (e: JSONException) {
            "Failed to parse prediction: ${e.message}"
        }
    }

    private fun sendImageToApi(fileUri: Uri) {
        val client = OkHttpClient()
        val apiUrl = "https://htrapi.onrender.com/process-image/"

        try {
            // Відкриваємо файл з URI та створюємо Bitmap
            val inputStream = contentResolver.openInputStream(fileUri)

```

```

val originalBitmap = BitmapFactory.decodeStream(inputStream)

// Масштабуємо зображення до ширини 1280
val resizedBitmap = resizeImage(originalBitmap, 1280)

// Перетворюємо Bitmap в байти
val byteArrayOutputStream = ByteArrayOutputStream()
resizedBitmap.compress(Bitmap.CompressFormat.JPEG, 85,
byteArrayOutputStream)
val fileBytes = byteArrayOutputStream.toByteArray()

val requestBody = MultipartBody.Builder()
    .setType(MultipartBody.FORM)
    .addFormDataPart(
        "file",
        "photo.jpeg",
        RequestBody.create(
            "image/jpeg".toMediaType(),
            fileBytes
        )
    )
    .build()

// Створюємо HTTP-запит
val request = Request.Builder()
    .url(apiUrl)
    .post(requestBody)
    .build()

// Виконуємо запит в окремому потоці
Thread {
    try {
        val response = client.newCall(request).execute()
        val responseBody = response.body?.string()

        if (response.isSuccessful && responseBody != null) {
            // Парсимо відповідь JSON
            val predictionText = parsePrediction(responseBody)

            runOnUiThread {
                recognizedTextView.text = predictionText
            }
        } else {
            runOnUiThread {
                recognizedTextView.text =
                    "Error: ${response.code} - ${response.message}"
            }
        }
    } catch (e: Exception) {
        runOnUiThread {
            recognizedTextView.text = "Request failed: ${e.message}"
        }
    }
}.start()
} catch (e: Exception) {
    recognizedTextView.text = "Failed to send image: ${e.message}"
}
}
}

```