**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО”**

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**До захисту допущено:**
**Завідувач кафедри**
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2023 р.

# Дипломний проєкт

**на здобуття ступеня бакалавра**

**за освітньо-професійною програмою “Комп’ютерні системи та мережі”**
**спеціальності 123 “Комп’ютерна інженерія”**

на тему: Система оптичного розпізнавання тексту

Виконала : студент __4__ курсу, групи _ІО-з91
(шифр групи)

_____Савенко Єлизавета Вареріївна_____
(прізвище, ім’я, по батькові) (підпис)

Керівник асистент кафедри ОТ, Кочура Юрій Петрович_____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) ст. викл. Виноградов Ю.М._____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент
_____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2023 р.

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО”**

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)
Освітньо-професійна програма
“Комп’ютерні системи та мережі”
спеціальності 123 “Комп’ютерна інженерія”

**ЗАТВЕРДЖУЮ**
**Завідувач кафедри**
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2023 р.

**ЗАВДАННЯ**
на бакалаврський дипломний проєкт студента

Савенко Єлизавети Валеріївни

1. Тема проєкту *Система оптичного розпізнавання тексту*
керівник проєкту асистент кафедри ОТ, Кочура Юрій Петрович
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

2. Термін здачі студентом закінченого проєкту *19 травня 2023 року*

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

4. Зміст роботи :
Розділ 1. Літературний огляд проблеми.
Розділ 2. Набір даних та ознаки для навчання.
Розділ 3. Деталі розробки системи.
Розділ 4. Експерименти та результати.

5. Перелік графічного матеріалу (з точним позначенням обов’язкових креслень): діаграма активності (структурна схема), діаграма класів (функціональна схема), діаграма потоку даних (принципова схема), текст програмного коду.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

| Розділ | Консультант | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| Нормоконтроль | Виноградов Ю.М. | | |
| | | | |

7. Дата видачі завдання «10» грудня 2023 року

Календарний план

| № п/п | Найменування етапів дипломного проєкту | Терміни виконання етапів проєкту | Примітки |
|---|---|---|---|
| 1. | *Затвердження теми проєкту* | 10.12.2022-15.12.2022 | |
| *2.* | *Вивчення та аналіз завдання* | 15.12.2022-20.03.2023 | |
| *3.* | *Програмна реалізація системи* | 20.03.2023-17.04.2023 | |
| 4. | *Оформлення пояснювальної записки* | 17.04.2023-21.05.2023 | |
| 5. | *Захист програмного продукту* | 21.05.2023-27.05.2023 | |
| *6.* | *Передзахист* | | |
| *7.* | *Захист* | 21.06.2023 | |

Студент _____ Єлизавета САВЕНКО
(підпис)

Керівник _____ Юрій КОЧУРА
(підпис)

# АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з 4 розділів, де в результаті розробляється система оптичного розпізнавання тексту. Попередньо розглядаються різні підходи для розв'язання цієї задачі, проводиться аналіз вже готових систем, а після пишеться власна система OCR, яка вбудована в бота для Telegram.

У першому розділі розглядаються можливі підходи і готові рішення цієї задачі. У другому представлено роботу з датасетом. У третьому описані деталі розробки системи та математичний підхід до всіх використаних підходів та алгоритмів. У четвертому надані експерименти та запровадження налаштованої моделі в готовий продукт у вигляді Telegram бота.

Система OCR для сегментації зображень використовує ручні ознаки, а модель розпізнавання у своїй основі має CNN архітектуру, яка навчалася на датасеті "UkrainianOCR".

Код написаний мовою Python із застосуванням бібіліотек keras і tensorflow для навчання моделі глибокого навчання. Для написання бота було використано бібліотеку aiogram.

Ключові слова: OCR, CNN, оптичне розпізнавання текста, deep learning, python

## ABSTRACT

The explanatory note of the thesis project consists of 4 chapters, where the optical text recognition system is developed. We first consider various approaches to solving this problem, analyze existing systems, and then write our own OCR system, which is built into a bot for Telegram.

The first section discusses possible approaches and existing solutions to this problem. The second section describes how to work with the dataset. The third section describes the details of the system development and the mathematical approach to all the approaches and algorithms used. The fourth section presents experiments and implementation of the customized model into a finished product in the form of a Telegram bot.

The OCR system for image segmentation uses manual features, and the recognition model is based on CNN architecture, which was trained on the UkrainianOCR dataset.

The code was written in Python using keras and tensorflow libraries to train the deep learning model. The aiogram library was used to write the bot.



Keywords: OCR, CNN, optical text recognition, deep learning, python

| справки | Формат | *Значення* | *Найменування* | Кіл. листів | № екземпля | *Додаток* |
|---|---|---|---|---|---|---|
| | | | Документація загальна | | | |
| | | | Знову розроблена | | | |
| | | | | | | |
| | ***А4*** | ***ІАЛЦ.467200.002 ТЗ*** | Система оптичного | 3 | | |
| | | | розпізнавання тексту | | | |
| | | | Технічне завдання | | | |
| | | | | | | |
| | | | | | | |
| | ***А4*** | ***ІАЛЦ.467200.003 ПЗ*** | Система оптичного | 80 | | |
| | | | розпізнавання тексту | | | |
| | | | Пояснювальна записка | | | |
| | | | | | | |
| | | | | | | |
| | ***А4*** | ***ІАЛЦ.467200.004 Д1*** | Система оптичного | 1 | | |
| | | | розпізнавання тексту | | | |
| | | | Діаграма класів | | | |
| | | | (функціональна схема) | | | |
| | | | | | | |
| | | | | | | |
| | ***А4*** | ***ІАЛЦ.4672008.005 Д2*** | Система оптичного | 1 | | |
| | | | розпізнавання тексту | | | |
| | | | Діаграма активності | | | |
| | | | (структурна схема) | | | |
| | | | | | | |
| | | | | | | |
| | ***А4*** | ***ІАЛЦ.4672008.006 Д3*** | Система оптичного | 1 | | |
| | | | розпізнавання тексту | | | |
| | | | Діаграма потоку даних | | | |
| | | | (принципова схема) | | | |
| | | | | | | |
| | | | | | | |
| | ***А4*** | ***ІАЛЦ.467200.007 Д4*** | Система оптичного | 20 | | |
| | | | розпізнавання тексту | | | |
| | | | Текст програмного коду | | | |
| | | | | | | |
| | | | | | | |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | ***ІАЛЦ.467200.001 ОА*** |
| *Зм* | *Лист* | *№ докум.* | *Підп* | *Дата* | |
| ***Розроб*** | | Савенко Є.В. | | | **Система оптичного розпізнавання тексту** Опис альбому |
| ***Перев*** | | Кочура Ю.П.. | | | |

| Літ. | Аркуш | Аркушів |
|---|---|---|
| | 1 | 1 |

КПІ ім. Ігоря Сікорського, ФІОТ, ІО-з91

# ТЕХНІЧНЕ ЗАВДАННЯ
# ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Система оптичного розпізнавання тексту»

Київ – 2023

# ЗМІСТ

# 1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку системи оптичного розпізнавання тексту, яка здатна обробляти скріншоти екрану з печатним текстом на українській мові.

Областю застосування цієї системи є широкий спектр сфер, де потрібне оцифрування тексту. Це може включати, але не обмежується, сферами як освіта, наукові дослідження, обробка документів, робота з архівами, бібліотеки, перекладачі та інші професійні галузі, де знадобляється введення тексту з печатних документів.

# 2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр інженерії програмного забезпечення», який був затверджений факультетом “Інформатики та обчислювальної техніки” кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

# 3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням даної роботи є розробка системи оптичного розпізнавання тексту із використанням сучасних алгоритмів, що дозволить ефективно вирішувати дану задачу.

# 4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного дипломного проекту є офіційні документації, публікації та статті в мережі Інтернет на дану тему, науково-технічна література.

# 5 ТЕХНІЧНІ ВИМОГИ

## 5.1. Вимоги до розробленого продукту

Розроблена система має виконувати такі вимоги:

- Простий і інтуїтивно-зрозумілий інтерфейс системи.
- Надати можливість користувачам передавати вхідні зображення тексту та отримувати у відповідь розпізнаний текст.
- Надати можливість користувачам власноруч вибирати бажаний алгоритм для розпізнавання тексту.
- Надати вичерпну та зрозумілу документацію для розробленого продукту.

## 5.2. Вимоги до програмного забезпечення

- ОС Windows, Mac чи Linux.
- PyCharm 2017 IDE версії або вище.

## 5.3. Вимоги до апаратної частини

- ЦП не менше ніж Intel® Core (TM) i3-2100T.
- ROM не менше ніж 64 ГБ.
- RAM не менше ніж 4 ГБ.

# 6 ЕТАПИ РОЗРОБКИ

| Назва етапів виконання | Термін виконання |
|---|---|
| Затвердження теми роботи | 10.12.2022-15.12.2022 |
| Вивчення та аналіз завдання | 15.12.2022-20.03.2023 |
| Розробка архітектури та загальної структури системи | 20.03.2023-26.03.2023 |
| Розробка структур окремих частин системи | 26.03.2023-6.04.2023 |
| Програмна реалізація системи | 6.04.2023-15.04.2023 |
| Виправлення помилок | 15.04.2023-17.04.2023 |
| Оформлення пояснювальної записки | 17.04.2023-21.05.2023 |

# Пояснювальна записка

# До дипломного проєкту

на тему: «Система оптичного розпізнавання тексту»

Київ – 2023

# ЗМІСТ

# СПИСОК СКОРОЧЕНЬ

OCR - Optical character recognition

CNN - Convolutional Neural Networks

RNN - Recurrent Neural Networks

YOLO - You Only Look Once

LSTM - Long Short-Term Memory

# ВСТУП

В епоху цифрової трансформації обробка та аналіз текстових даних стали важливим кроком у багатьох галузях промисловості та повсякденному житті. Однак не всі дані можна легко обробити в цифровому форматі. Велика кількість інформації все ще зберігається у вигляді рукописних або друкованих документів, які потребують оцифрування. Саме тому ці системи набувають все більшої популярності.

Застосування технологій OCR має широкий спектр у сучасному світі, починаючи від сканування та конвертації документів, і закінчуючи зчитуванням тексту для людей з вадами зору. Однак, враховуючи широке розмаїття шрифтів, стилів написання та форматів документів, а також пошкоджені або неякісні зображеня, розробка ефективної системи оптичного розпізнавання стає складним завданням.

Далі в цій роботі буде розглянуто різні підходи, які допомагають уникати різних складнощів під час розпізнавання тексту.

# РОЗДІЛ 1 ЛІТЕРАТУРНИЙ ОГЛЯД ПРОБЛЕМИ

## 1.1. Аналіз актуальності

Розпізнавання тексту в цифровий формат набуває популярності та відіграє важливу роль у сучасному суспільстві. Основні досягнення в цій галузі були отримані завдяки машинному навчанню й активному розвитку технологій.

Використання технології OCR дає змогу розв'язувати численні завдання, пов'язані з вилученням і опрацюванням інформації з друкованих і рукописних текстів. Основні сфери застосування OCR включають:

- архівування та оцифрування документів. Це забезпечує легкий доступ, пошук і зберігання інформації. Особливо актуально для державних установ, бібліотек, музеїв і дослідницьких організацій

- обробка опитувань і форм в автоматичному режимі

- розпізнавання тексту для людей з обмеженими можливостями, а потім подальшого синтезу мови, забезпечуючи можливість людям з порушенням зору комфортно комунікувати із зовнішнім світом

- розпізнавання тексту на відео. Дає змогу автоматично записувати метадані, покращуючи тим самим пошук контенту

- переклад тексту в режимі реального часу. Дозволяє подолати будь-який мовний бар'єр

- управління банківськими операціями шляхом автоматичного опрацювання чеків і рахунків

- контроль і безпека на дорогах шляхом розпізнавання номерних знаків і документів

Технологія OCR сприяє вирішенню проблем, які пов'язані з мовними бар'єрами, фізичними труднощами або доступністю, розширюючи таким чином доступ до знань для всіх. Одним із важливих аспектів є продовження інвестування в розробку технології OCR та вдосконалення діючих алгоритмів з метою підвищення точності та продуктивності в перспективі.

Таким чином, оптичне розпізнавання тексту не тільки підтверджує свою актуальність, а й продовжує надавати нові можливості для різноманітних галузей і сфер діяльності.

## 1.2. Аналіз істуючих підходів

Мета оптичного розпізнавання символів - перетворити зображення тексту у формат, зручний для машинного читання. Зазвичай процес поділяється на два ключових етапи: пошук тексту та його подальше розпізнавання (рис. 1.1.).

Пошук тексту - це процес ідентифікації областей зображення, які містять текст. Цей етап може бути доволі складним, оскільки текст може бути розташований у будь-якій частині зображення, може мати різні розміри, шрифти та кольори, а також може бути поверненим або спотвореним. Крім того, текст може перебувати на різних фонах, які можуть ускладнити його виявлення.

Розв'язання цієї задачі зазвичай досягається за допомогою численних методів, включаючи як ручні, так і засновані на машинному навчанні. Широке застосування знаходять згорткові нейронні мережі (CNN), які здатні розпізнавати структури і патерни, характерні для тексту на зображеннях.

Після того, як області з текстом виявлено, настає етап розпізнавання тексту. Він передбачає визначення кожного символу в знайдених областях і перетворення його у формат, придатний для машинного читання. Це також складне завдання, оскільки кожен символ може бути написаний різними

шрифтами, може бути спотворений, повернутий, мати різний розмір і стиль. Окремо варто виділити складність розпізнавання рукописного тексту, який може суттєво відрізнятися від друкованого, що робить завдання ще складнішим. Вирішення цього завдання зазвичай досягається за допомогою рекурентних нейронних мереж (RNN), здатних обробляти послідовності даних.

Останній етап включає в себе використання мовної моделі для корекції помилок OCR. Це важливо, оскільки на етапі розпізнавання символів можуть виникнути помилки через шум, спотворення, нестандартні шрифти тощо. Мовні моделі можуть допомогти поліпшити точність, передбачаючи найімовірніші символи або послідовності символів на основі контексту та статистичної інформації про мову.

Але так само існує й інший підхід, який не розділяє завдання на два етапи, а сприймає його як безперервний процес. Це так звані енд-ту-енд (end-to-end) моделі, що приймають на вхід зображення і видають на виході розпізнаний текст, без чіткого поділу на етапи пошуку і розпізнавання тексту. Ці моделі навчаються на парах "зображення-текст" і самостійно визначають, які елементи зображення ключові для розпізнавання тексту.

Такі моделі можуть бути засновані на різних типах нейронних мереж, включно зі згортковими, рекурентними нейронними мережами і трансформерами. Навчання таких моделей може бути складнішим, оскільки вони повинні справлятися з більш трудомістким завданням, але водночас вони можуть бути більш ефективними. Причина ефективності полягає в тому, що ці моделі можуть вчитися використовувати всю інформацію зображення для більш точного розпізнавання тексту.

Також важливим етапом є попередня обробка даних. Вона може значно поліпшити підсумкові дані. Попереднє опрацювання може містити безліч різних методів і технік, залежно від специфіки даних і вимог до моделі.

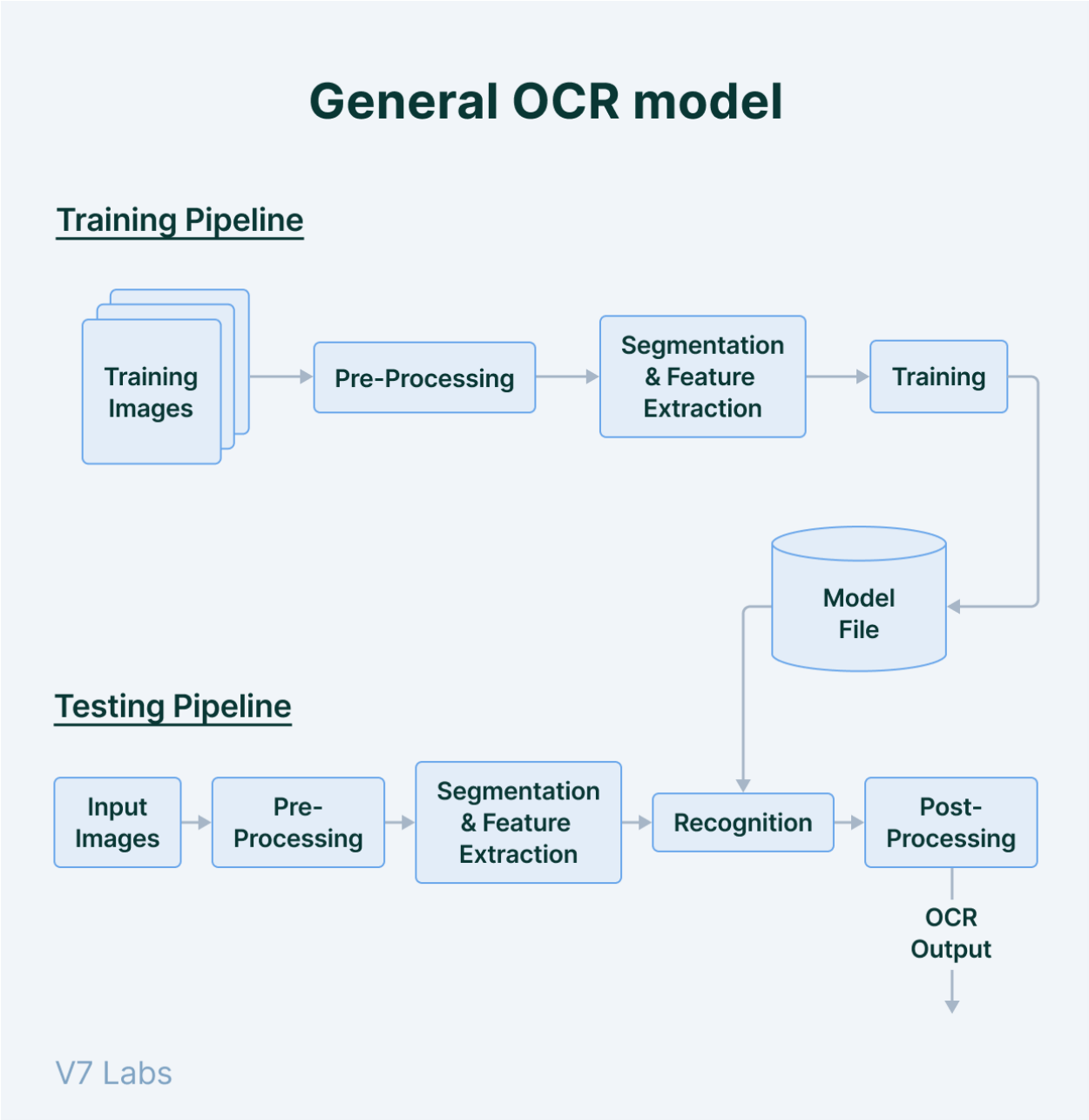



Рисунок 1.1. – Стуктура типової OCR системи [1]

## 1.3. Методи сегментації

Умовно, методи можна розділити на 2 частини:

1) Методи на основі глибокого навчання: ці методи використовують різні архітектури глибокого навчання, такі як CNN, R-CNN, Fast R-CNN, та інші для виявлення тексту на зображеннях. Ці методи

можуть навчатися на великих обсягах даних і добре справляються з виявленням тексту в складних умовах.

2) Так само можна використовувати модель YOLO (You Only Look Once), яка є однією з найефективніших моделей для виявлення об'єктів у реальному часі і може бути адаптована для виявлення тексту. Основна відмінність YOLO від інших методів виявлення об'єктів полягає в тому, що він обробляє все зображення за один раз, що робить його швидким і придатним для реального часу.

Методи на основі ручних ознак: ці методи включають у себе:

- Методи на основі порогових значень
- Використання шаблонів і матриць зіставлення
- Геометричні та статистичні ознаки
- Морфологічний аналіз

Та інші. Більш детально ці методи розглядатимуться у 3 розділі.

## 1.4. Методи розпізнавання

На цьому етапі варіати підходів ширші:

1) Рекурентні нейронні мережі (RNN): Це один із найпоширеніших підходів для обробки тексту після його виявлення. Зокрема, варіації RNN, такі як LSTM (довгострокова короткострокова пам'ять) або GRU (оновлювані одиниці із загасанням), часто використовують для цього завдання, оскільки вони добре справляються з послідовностями даних, такими як текст.

2) Трансформери: Трансформери - це тип архітектури нейронних мереж, заснований на механізмі уваги, що дає змогу моделі сконцентруватися на різних частинах вхідних даних залежно від їхньої важливості для конкретного завдання. У контексті розпізнавання тексту трансформери можуть ефективно обробляти текстові послідовності, враховуючи контекст кожного символу або слова, що призводить до більш точного і гнучкого розпізнавання тексту.

3) Методи на основі згорткових нейронних мереж (CNN): CNN здатні виділяти ієрархічні ознаки з вхідних даних, починаючи від простих і закінчуючи складними. У разі опрацювання тексту, CNN можуть використовуватися для розпізнавання символів або слів, виявляючи форми та візерунки, що забезпечує їхню здатність адаптуватися до різних стилів і шрифтів.

## 1.5. Методи корекції помилок

Під час вилучення інформації із зображень часто виникають помилки або неточності в процесі розпізнавання. Нижче представлено кілька підходів до корекції помилок:

1) Використання мовних моделей: Мовні моделі, засновані на статистичних методах або на глибокому навчанні, як-от n-gram або transformers, можна використовувати для виправлення помилок. Вони можуть передбачити найімовірніше наступне слово або символ на основі контексту. Це може бути корисно для корекції помилок, заснованих на контекстній інформації.

2) Spell-checking алгоритми: Алгоритми перевірки орфографії, наприклад, метод Левенштейна (Levenshtein distance) або методи, засновані на Fuzzy matching.
3) Sequence-to-sequence моделі: Моделі типу sequence-to-sequence (seq2seq), засновані на RNN, LSTM, або Transformer мережах, можуть бути навчені для виправлення помилок. Вони можуть навчатися на парах прикладів "вихідний текст - виправлений текст" і використовуватися для виправлення помилок у нових даних.
4) Ансамблювання моделей: Використання ансамблів моделей може допомогти знизити кількість помилок. Кілька моделей навчаються незалежно, а потім їхні прогнози комбінуються (часто шляхом голосування), щоб отримати остаточний результат. Якщо одна модель припускається помилки, інші моделі можуть її компенсувати.

## 1.6. Методи енд-ту-енд (end-to-end)

Енд-ту-енд моделі створені таким чином, щоб вони могли виконувати всю послідовність завдань від початку до кінця без явних проміжних кроків, при цьому всі частини процесу оптимізуються разом. У контексті оптичного розпізнавання тексту, це означає, що енд-ту-енд модель може одночасно виконувати виявлення тексту, сегментацію тексту і розпізнавання тексту.

Однією з важливих переваг такого підходу є те, що модель може вчиться від великого масиву вхідних данних і використовувати цю інформацію для більш точного розпізнавання тексту. Більше того, енд-ту-енд підхід може забезпечити більшу ефективність, оскільки він не потребує великої кількості ручної роботи або додаткових кроків для обробки данних. Однак вони можуть бути важкими в тренуванні і потребують значного обсягу данних для ефективного навчання.

Приклади таких моделей:

1) EAST (Efficient and Accurate Scene Text Detector): Це енд-ту-енд модель, яка здатна виявляти текст у природних сценах. Вона може одночасно виявляти і розпізнавати текст, що робить її дуже ефективною.
2) YOLO: Хоча YOLO зазвичай використовується для виявлення об'єктів (як було описано вище), його також можна використовувати для виявлення тексту в реальному часі.
3) Підходи на основі CRNN: CRNN, або згортково-рекурентні нейронні мережі, об'єднують переваги CNN і RNN. У цьому підході згорткові шари спочатку використовуються для вилучення ознак із зображень. Потім рекурентні шари інтерпретують ці ознаки як послідовності для розпізнавання тексту. Це забезпечує CRNN здатність обробляти текст з урахуванням його просторової структури і послідовної природи, що важливо для ефективного розпізнавання тексту.

## 1.7. Порівняльний аналіз наявних методів OCR

### 1.7.1. Google Cloud Vision OCR

Google Cloud Vision OCR є частиною Google Cloud Platform, яка була розроблена в 2016 році. Технологія заснована на комп'ютерному зорі та машинному навчанні, розроблена самою компанією Google.

Як сказано на самому сайті:

"Cloud Vision дозволяє розробникам легко інтегрувати функції розпізнавання зорового сприйняття в додатки, включаючи маркування зображень, розпізнавання облич і орієнтирів, оптичне розпізнавання символів (OCR) і тегування явного контенту"[2].

Для використання Google Cloud Platform ( далі GCP) OCR необхідно створити обліковий запис на GCP і ввімкнути API Vision. Потім можна використовувати надані бібліотеки та сервіси для подальшої інтеграції з додатками.

### 1.7.2. Microsoft Azure Cognitive Services OCR

Microsoft Azure Cognitive Services OCR - це сервіс хмарної інтелектуальної обробки зображень, який вміє розпізнавати текст з різноманітних зображень і документів. Microsoft Azure Cognitive Services OCR (MCS) є частиною Microsoft Azure Cognitive Services, який, своєю чергою, являє собою набір інтелектуальних сервісів для аналізу даних. Цю систему OCR було розроблено 2016 році.

Він використовується для автоматизації розпізнавання тексту, екстракції інформації та обробки даних. Використовуючи передові алгоритми машинного навчання, сервіс може розпізнавати текст в багатьох мовах, в тому числі і рукописний текст, з великою точністю і швидкістю. Крім того, Azure OCR може адаптуватися до ваших вимог, навчаючись від своїх власних помилок та усовершенствуючись з часом. Це робить його відмінним інструментом для багатьох застосувань, включаючи автоматизоване введення даних, аналіз контенту та сортування документів.

### 1.7.3. ABBYY FineReader

ABBYY FineReader - це програма для оптичного розпізнавання символів (OCR), яка використовує передові технології для перетворення зображень, PDF файлів та інших сканованих документів в редаговані і пошукові формати, такі як Microsoft Word, Excel, PowerPoint, або у формати електронних книг. Першу версію FineReader було випущено 1993 року. Система досі постійно розвивається для задоволення вимог ринку, що змінюються. Наразі FineReader підтримує 201 мову розпізнавання.

FineReader здатний визначати мову тексту, структуру документа, таблиці, шрифти і стилі форматування, а також відтворює логічну структуру документів при конвертації. Додаток також здатний розпізнавати текст, що міститься на зображеннях і в PDF файлах, що дозволяє проводити повнотекстовий пошук по цим документам. За допомогою технології Adaptive Document Recognition Technology (ADRT), ABBYY FineReader забезпечує більш точне форматування та стилізацію, включаючи візуальні елементи, як-то картинки, заголовки, футери, нумерацію сторінок і заголовки таблиць. Дерево поділу тегнологій зображено на рис. 1.2.

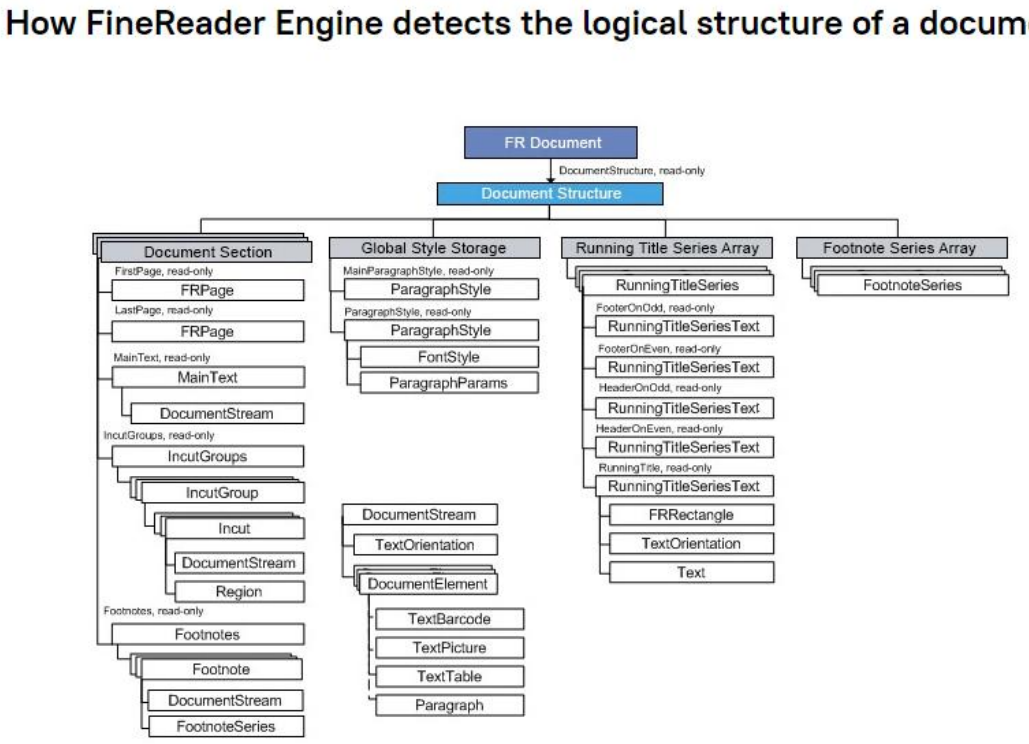



Рисунок 1.2. – Принцип поділу технології ADRT[3]

### 1.7.4. Readiris

Readiris - це високоефективний інструмент для оптичного розпізнавання символів (OCR), який підтримує перетворення паперових

документів, PDF-файлів та зображень у редаговані й пошукові формати. Він відтворює документи з точним збереженням оригінального форматування, включаючи графіку, колонки тексту, колонтитули, нумерацію сторінок та таблиці. Readiris підтримує багато мов і може автоматично визначити мову тексту. Крім того, програма має можливість розпізнавання голосу для аудіо-транскрипції, можливість перетворення документів у електронні книги і високую ступінь інтеграції з хмарними сервісами.

### 1.7.5. Adobe Acrobat Pro

Adobe Acrobat Pro - це потужний інструмент для створення, редагування, перегляду і конвертування PDF-файлів. Він включає в себе функцію OCR, яка дозволяє користувачам конвертувати скановані документи, зображення або PDF-файли, які були створені з зображень, в пошукові і редаговані PDF-файли.

Adobe Acrobat Pro відмінно підходить для інтеграції з іншими продуктами Adobe та має потужні інструменти редагування PDF. Він підтримує багато мов, може обробляти складні макети та є надійним рішенням для професійних потреб.

# ВИСНОВОК ДО РОЗДІЛУ 1

В першому розділі були розглянуті  різних методі та підходи до OCR, що включають сегментацію, розпізнавання, корекцію помилок і енд-ту-енд методи.

Ми встановили, що незважаючи на велику кількість доступних рішень на ринку, включаючи Google Cloud Vision OCR, Microsoft Azure Cognitive Services OCR, ABBYY FineReader, Readiris, та Adobe Acrobat Pro, немає універсального рішення, що було б оптимальним для всіх сценаріїв. Вибір специфічного методу або інструменту значною мірою залежить від потреби користувача та специфіки завдання.

# РОЗДІЛ 2 НАБІР ДАНИХ ТА ОЗНАКИ ДЛЯ НАВЧАННЯ

## 2.1. Огляд даних

Цей підрозділ містить огляд і аналіз датасетів для навчання моделі. Набір даних потрібен для навчання моделі для розпізнавання тексту на вже сегментованих даних.

## 2.2. Український датасет

Для навчання моделі глибокого навчання за основу буде взято датасет uaset (рис. 2.1.) за зразком класичного EMNIST, але автоматично згенерований для друкованих літер української мови.


Рисунок 2.1. - Унікальні мітки з датасету uaset з прикладом зображення

Набір даних uaset або ж "UkrainianOCR", містить 9966 зображень, розподілених на 66 класів. Кожен клас відповідає окремому символу української мови в маленькому і великому регістрі.

Структура датасету представлена у форматі CSV, де кожен рядок відповідає окремому зразку для тренування, де:

Мітка - перше число кожного рядка в діапазоні від 0 до 65, що відповідає конкретному символу української мови.

Зображення - наступні 784 числа в діапазоні від 0 до 255, які представляють собою 8-бітне зображення в градаціях сірого розміром 28x28 пікселів [4].

Символи в наборі розподілені рівномірно, як видно на рис.2.2.

У процесі тренування моделі розміри зображень зберігаються без змін, оскільки це є оптимальним розміром для розпізнавання символів за допомогою глибокої згорткової нейронної мережі. Детальний аналіз та опис класів, розподілу зображень по класах, а також технік візуалізації та передпроцесингу даних будуть представлені в наступних розділах.

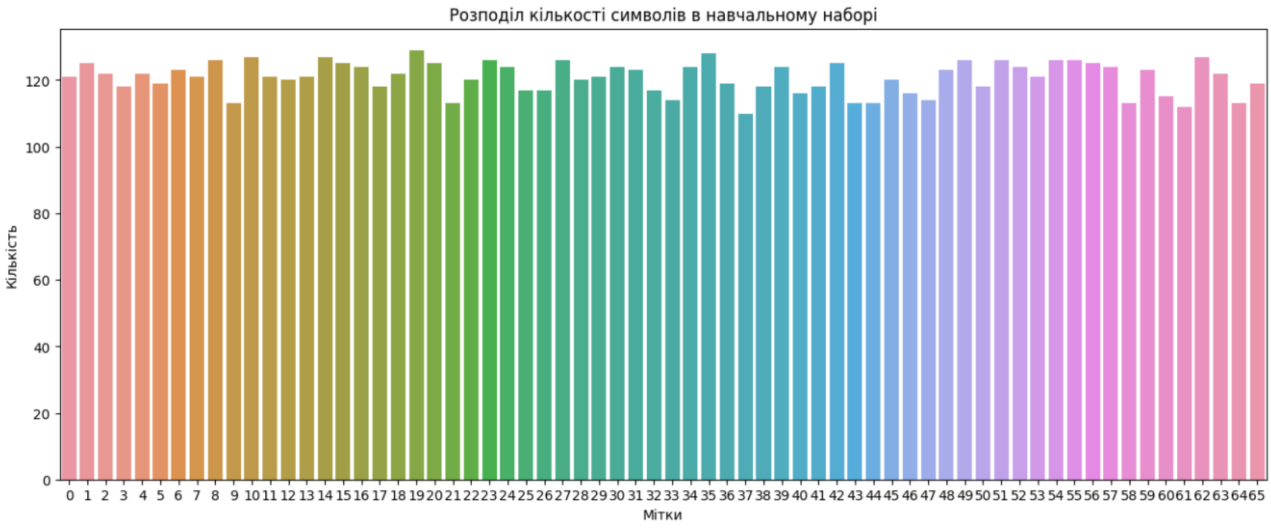



Рисунок 2.2. – Розподіл кількості символів в навчальному наборі

### 2.2.1. Покращення датасету

Початковий датасет у своєму наборі даних містив зображення символів, розташованих не по ширині зображення, а в трохи меншому форматі, що утворювало великий простір навколо символу. Під час подальшої сегментації тексту, зображення буде обрізатися якраз так, щоб символи були по ширині зображення, отже, аналогічно мають виглядати і навчальні дані як на рис.2.3.


Рисунок 2.3. – Обрізані зображення з датасету

### 2.2.2. Аугментація даних

Аугментація даних є однією з ключових стратегій, яка допомагає покращити ефективність моделей глибокого навчання, особливо в задачах комп'ютерного зору. Вона полягає в створенні нових зразків тренувальних даних шляхом застосування різноманітних трансформацій до оригінальних зображень, таких як перевертання, зміна масштабу, обертання, зсув та інші види перетворень.

В контексті нашого проекту, аугментація даних буде використовуватися для розширення набору даних. Це дозволить збільшити кількість

тренувальних даних та ще допоможе моделі краще навчитись розпізнавати символи в різноманітних умовах.

Така стратегія важлива, оскільки в реальному світі зображення символів можуть відрізнятися за розміром, орієнтацією, масштабом і можуть бути спотворені різними факторами.

Реалізацію аугментації було виконано за допомогою вбудованого класу ImageDataGenerator з keras. Ми застосували аугментацію, що включає зум з діапазоном 0.1 та вертикальний зсув з діапазоном 0.1.

Також додали гаусовий шум до наших зображень зі стандартним відхиленням 0.1. Для цього було необхідно написати власний клас для додавання шуму й інтегрували його в ImageDataGenerator, оскільки це не входить у можливості стандартного класу.

Приклад аугментованих зображень можна побачити на рис. 2.4.

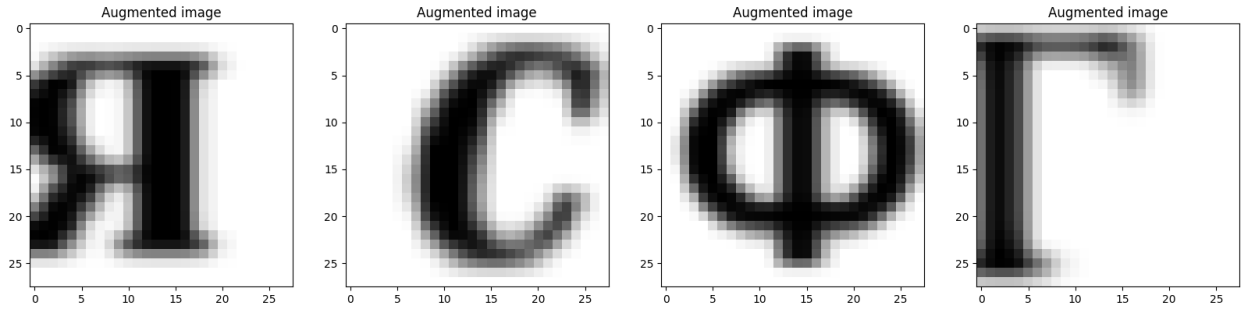



Рисунок 2.4. – Приклад аугментованих зображень

### 2.2.3. Поділ датасету

Для того, щоб наша модель була здатна відповідно оцінювати свою продуктивність і адекватно покращуватись під час навчання, ми розділили наші дані на тренувальний та тестовий набори за допомогою функції train_test_split з бібліотеки sklearn.model_selection, встановивши розмір тестового набору як 20% від загального набору даних. До нормалізації даних ми поділили усі значення признаків на 255.0, змінюючи діапазон з 0-255 до 0-

1, а потім змінили його на протилежний за допомогою 1 - X, щоб повернути кольори зображення, так як вихідні дані містили зображення з інвертованими кольорами.

Наші зображення мають розмір 28x28 пікселів, тому ми переформували вхідні дані для конволюційної нейронної мережі до форми (-1, 28, 28, 1), де '-1' вказує на те, що кількість зразків буде визначатися автоматично на основі розміру набору даних. Крім того, ми конвертували наші мітки до векторів 'one-hot' за допомогою функції to_categorical з бібліотеки keras.utils, щоб забезпечити їх сумісність з вимогами вихідного шару нашої мережі.

# ВИСНОВОК ДО РОЗДІЛУ 2

В основу навчання моделі глибокого навчання було взято датасет "UkrainianOCR", який складається з 9966 зображень, що розподілені на 66 класів. Кожен клас відповідає окремому символу української мови в маленькому і великому регістрі. Щоб покращити якість набору даних, було проведено аугментацію, включаючи масштабування, вертикальний зсув та додавання гаусового шуму.

Далі набір даних було розділено на тренувальний та тестовий набори, використовуючи функцію train_test_split з бібліотеки sklearn.model_selection, де розмір тестового набору становить 20% від загального обсягу даних. Для нормалізації даних було виконано ділення всіх значень признаків на 255.0, а потім проведено інверсію кольорів.

Перед подачею в модель, вхідні дані були переформатовані до вимог конволюційної нейронної мережі, а мітки були конвертовані в one-hot вектори. Цей підхід дозволив підготувати оптимальний набір даних для тренування та оцінювання моделі, що сприяло покращенню її продуктивності.

# РОЗДІЛ 3 ДЕТАЛІ РОЗРОБКИ СИСТЕМИ

## 3.1. Вибір мови програмування та бібліотек

### 3.1.1. Python

Python - це універсальна високорівнева мова програмування, яка наразі є однією з найпопулярніших мов для наукових обчислень, аналізу даних і, безумовно, машинного навчання, зокрема й для завдань OCR.

Основними причинами вибору Python у контексті OCR є багата екосистема бібліотек, яка містить у собі TensorFlow, PyTorch, Keras, OpenCV, PIL/Pillow, Scikit-learn і багато інших, що значно полегшує завдання машинного навчання й опрацювання зображень. Так само Python має величезну спільноту розробників, яка регулярно оновлює наявні бібліотеки та створює нові.

Вибір Python зумовлений низкою зазначених вище причин. У наступних розділах буде представлено опис ключових бібліотек, які використовують під час розв'язання задач OCR.

### 3.1.2. TensorFlow

TensorFlow - це відкрита платформа для машинного навчання, розроблена Google. Вона пропонує великі функціональні можливості та інструменти, які роблять її придатною для різних завдань.

TensorFlow може бути використана для навчання і розгортання глибоких нейронних мереж. Однією з переваг TensorFlow є її гнучкість, яка дає змогу розробникам створювати свої власні моделі або адаптувати наявні.

За допомогою TensorFlow можна не тільки навчати моделі на CPU, а й використовувати більш потужні GPU для прискорення процесу навчання.

Крім того, бібіліотека пропонує інструменти для візуалізації процесу навчання і налагодження моделей, такі як TensorBoard.

Бібліотека TensorFlow сумісна з Python, що робить її чудовим вибором для проєктів, пов'язаних з OCR, однак також вона підтримує інші мови, такі як C++, Java і JavaScript.

### 3.1.3. Matplotlib

Matplotlib - це бібліотека Python, призначена для створення графіків і діаграм. Вона має значну гнучкість і потужні можливості для створення різноманітних візуалізацій, що робить її невід'ємним інструментом для дослідників даних і спеціалістів у галузі машинного навчання.

У контексті OCR і цієї роботи, Matplotlib може бути використана для візуалізації даних перед початком роботи над моделлю машинного навчання, що допомагає краще зрозуміти характеристики та складності даних. Під час навчання моделі, Matplotlib може бути використана для відображення зміни значень функції втрат і точності моделі протягом кожної епохи, допомагаючи відслідковувати прогрес навчання і виявляти можливі проблеми, такі як перенавчання. Після завершення навчання моделі, Matplotlib можна використовувати для візуалізації результатів, таких як відображення матриці помилок або прикладів правильних і неправильних передбачень.

Так само ми будемо використовувати поліпшену версію matplotlib - seaborn. Ця бібліотека має більш розширений функціонал для візуалізації графіків.

### 3.1.4. OpenCV

OpenCV (Open Source Computer Vision) - це потужна бібліотека для роботи з комп'ютерним зором і машинним навчанням. Вона пропонує різноманітні алгоритми та інструменти для аналізу зображень і відео. У контексті OCR, OpenCV особливо корисна для попереднього опрацювання зображень: перетворення колірного простору, порогового опрацювання,

масштабування і вирівнювання зображень та інших морфологічних операцій. Це дає змогу ефективно підготувати зображення для подальшого аналізу за допомогою OCR-рушіїв. OpenCV написана на C++, але надає інтерфейси для Python, Java і MATLAB/OCTAVE, що робить її доступною для широкого кола розробників.

### 3.1.5. Онлайн-середовище розробки Google Colab

Google Colab (або Colaboratory) — це безкоштовне онлайн-середовище для розробки, яке базується на облачній технології Google. Воно дозволяє користувачам писати та запускати скрипти Python безпосередньо у веб-браузері, що полегшує роботу з аналізом даних та машинним навчанням.

Основна перевага Google Colab полягає в тому, що ви можете використовувати ресурси Google для виконання важких обчислювальних задач, таких як GPU (графічні процесори) та TPU (тензорні процесори), які значно прискорюють процеси машинного навчання та глибокого навчання.

Блокноти Google Colab підтримують інтерактивний код Python. Код розбивається на окремі "клітинки", які можна виконувати незалежно одна від одної. Це забезпечує велику гнучкість і дозволяє експериментувати з кодом в реальному часі.

У цій роботі ми використовуємо його для навчання і гнучкого налаштування моделі та візуалізації даних.

### 3.1.6. Aiogram

Aiogram - це сучасна, високопродуктивна, асинхронна бібліотека для розробки ботів на платформі Telegram, написана на Python. Ця бібліотека була використана в цьому проєкті для створення бота, здатного надавати візуалізацію результатів моделі оптичного розпізнавання символів. Aiogram забезпечує обробку всіх типів оновлень, пропонованих API Telegram. Завдяки його асинхронності та потужним інструментам для опрацювання повідомлень, бот, розроблений на основі Aiogram, забезпечує зручний

інтерфейс користувача, даючи змогу завантажувати зображення для розпізнавання символів і отримувати зворотний зв'язок у реальному часі.

Бібліотека Aiogram надає переваги, такі як створення інлайн та звичайних клавіатур для інтерактивної взаємодії з користувачами. Крім того, асинхронність Aiogram дає змогу боту обробляти кілька запитів одночасно, збільшуючи його продуктивність. Використання Aiogram у проєкті забезпечило створення високопродуктивного бота зі зручним інтерфейсом для візуалізації результатів роботи моделі оптичного розпізнавання символів.

## 3.2. Preprocessing

### 3.2.1. Важливість попередньої обробки даних

Попередня обробка (preprocessing) даних є невід'ємною частиною процесу розпізнавання тексту і відіграє ключову роль у забезпеченні високої точності та ефективності системи OCR.

Основна мета її - поліпшення якості вхідних даних і підготовка для обробки моделями. Це передбачає усунення шумів, нормалізацію, сегментацію та інші процеси, які полегшують подальше опрацювання та інтерпретацію даних моделями OCR. Ці методи покращують якість даних, видаляючи шуми та інші візуальні перешкоди, які можуть призвести до помилок під час розпізнавання символів. Коректне оброблення даних може значно поліпшити ефективність і точність моделей OCR і спростити саме розпізнавання.

### 3.2.2. Нормалізація

Нормалізація (Normalization) зображення в OCR - це процес, під час якого змінюється діапазон значень інтенсивності пікселів. Це робиться для приведення зображення до "нормального" діапазону, що робить його легше підданим обробці для наступних кроків алгоритму OCR[5].

У контексті OCR, нормалізація часто відноситься до приведення значень інтенсивності пікселів до певного діапазону, зазвичай від 0 до 1 або від 0 до 255. Це може спростити порівняння зображень і допомогти забезпечити більш надійне розпізнавання тексту. Вона виконується за формулою:

$$I_N = (I - Min)\frac{newMax - newMin}{Max - Min} + newMin \quad (3.1)$$

Де $I_N$ – зображення,

Min, Max – діапазон старих значень зображення

newMin, newMax - новий діапазон

Важливо пам'ятати, що хоча нормалізація може значно поліпшити якість розпізнавання OCR, вона не завжди є необхідною або корисною. Ефективність нормалізації залежить від характеристик конкретного зображення і використовуваного алгоритму OCR.

### 3.2.3.Вирівнювання (Deskew)

Вирівнювання (Deskew) відіграє важливу роль у процесі OCR. Воно застосовується для корекції зображень, спотворених через нахил або поворот. Коли документи скануються або фотографуються, сторінки можуть бути нахилені або спотворені. Це може зробити лінії тексту косими.

Процес вирівнювання починається з визначення кута нахилу. Це зазвичай досягається шляхом аналізу гістограм проекції, де осі X і Y аналізуються на предмет спотворень. Потім, після того як кут нахилу

визначено, зображення обертається на відповідний кут у зворотному напрямку, щоб повернути текст у горизонтальне положення.

Кут нахилу (θ) може бути визначений як:

$$\theta = \arctan\frac{(y_1 - y_0)}{(x_1 - x_0)} \tag{3.2}$$

де (x0, y0) є центром маси гістограми, а (x1, y1) є координатами будь-якої іншої точки на гістограмі.

Після визначення кута нахилу, зображення обертається на відповідний кут у зворотному напрямку. Це здійснюється за допомогою афінної трансформації, вираженої через матрицю обертання:

$$M = \begin{matrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{matrix} \tag{3.3}$$

Застосування цієї матриці до всіх пікселів зображення приводить до корекції нахилу та повороту, повертаючи текст до його первинного горизонтального положення.

Побачити приклад вирівнювання можна на рис. 3.1.

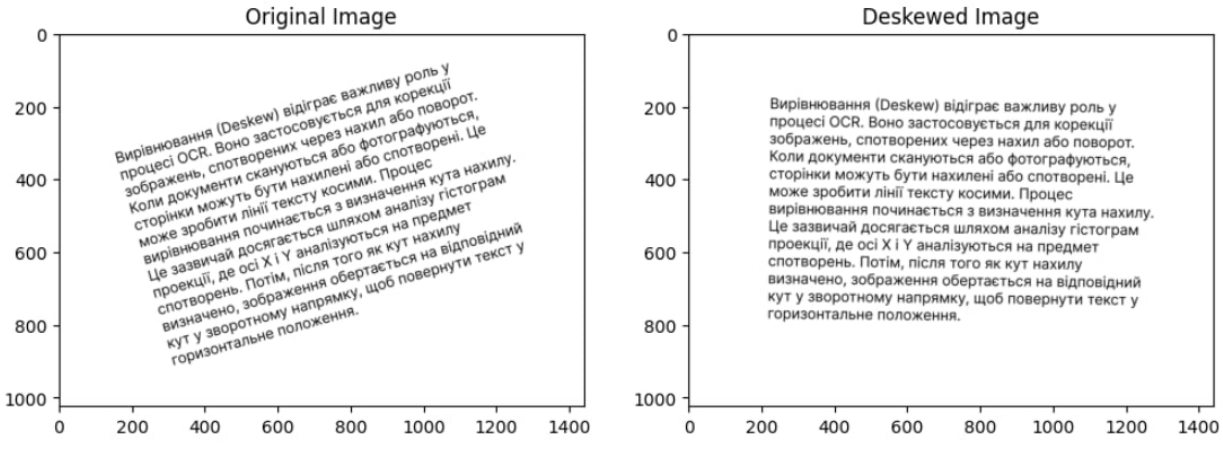



Рисунок 3.1. – Приклад вирівнювання

### 3.2.4. Масштабування зображення (Image Scaling)

Масштабування зображення - це коригування розміру зображення зі збереженням пропорцій. Це може означати як збільшення, так і зменшення ступеня деталізації цифрового зображення.

Масштабування зображення може бути виконано з використанням різних алгоритмів, кожен з яких має свої особливості та підходить для певних завдань. Наприклад, білінійне масштабування зазвичай добре працює для збільшення розміру зображень, але може спричинити втрату деталей під час зменшення розміру. Кубічне масштабування, з іншого боку, забезпечує вищу якість під час зменшення розміру зображення, але вимагає більше обчислювальних ресурсів.

Розглянемо формулу білінійної інтерполяції:

$$f_{(x,y)} = f_{(0,0)}(1-x)(1-y) + f_{(1,0)}x(1-y) + f_{(0,1)}(1-x)y + f_{(1,1)}xy \quad (3.4)$$

де:

$f_{(x,y)}$ – нове значення пікселя в точці (x, y)

$f_{(0,0)}$, $f_{(0,1)}$, $f_{(1,0)}$, $f_{(1,1)}$– значення найближчих пікселів до точки (x, y) на вихідному зображенні

x, y – відстані від точки до чотирьох найближчих пікселів

Білінійна інтерполяція дозволяє обчислити значення пікселя на вихідному зображенні на основі значень найближчих пікселів на вихідному зображенні, тим самим змінюючи розмір зображення.

### 3.2.5. Видалення шумів (Noise Removal)

Мета видалення шуму - мінімізувати або повністю усунути перешкоди для поліпшення якості вилучення тексту. Шум на зображенні може набувати різних форм: від випадкових пікселів різного кольору до складніших артефактів, таких як нерівності або плями на фотографії. Шум може бути спричинений різними факторами, включно з умовами зйомки, якістю камери або процесом сканування документів.

Видалення може бути досягнуто за допомогою різних технік і алгоритмів, включно з морфологічними операціями (наприклад, ерозією, розширенням, відкриттям і закриттям), згладжуванням (наприклад, розмиттям Гаусса, як на 3.2, або медіанним фільтром) та іншими спеціалізованими методами, такими як усунення шуму на основі статистичних методів.

Розглянемо як приклад метод розмиттям Гаусса або як його називають ще фільтр Гаусса.

Він використовується для усунення високочастотного шуму в зображеннях. Цей фільтр оснований на Гауссовій функції.

Математично, Гауссова функція в двовимірному просторі визначається наступною формулою:

$$g(x,y) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/(2\sigma^2)} \quad (3.5)$$

де σ – стандартне відхилення

x, y – координати пікселя

Фільтр Гаусса вимагає створення ядра (матриці), що представляє Гауссову функцію. Розмір ядра і величина σ обираються на основі бажаної ступені згладжування.

Після створення ядра воно перетинається з зображенням, при чому кожний піксель замінюється взваженим середнім його сусідів, ваги яких визначаються значеннями в ядрі.

Цей процес змінює значення кожного пікселя таким чином, що високочастотний шум згладжується, а низькочастотна інформація (така як краї та контури) зберігається.

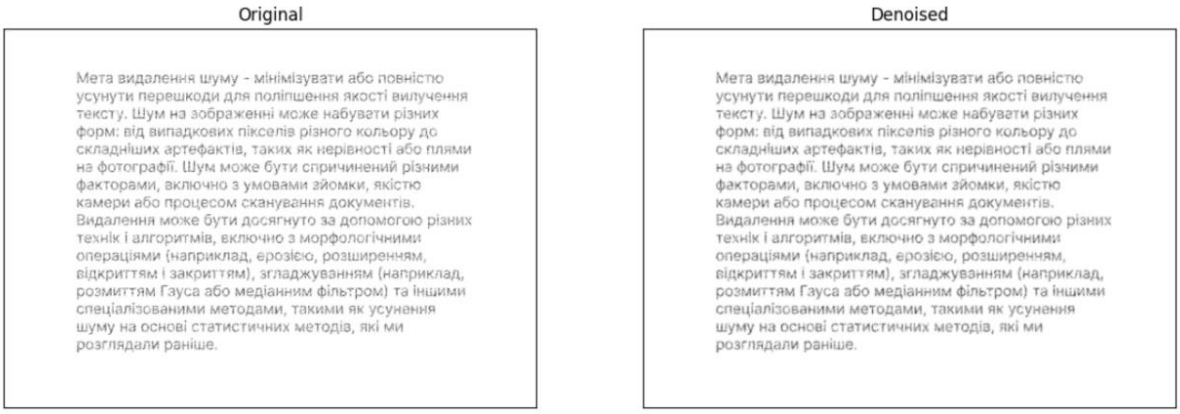



Рисунок 3.2. – Приклад видалення шуму

### 3.2.6. Скелетизація

Скелетизація - це процес обробки зображення, який дає змогу зменшити області переднього плану до скелетного залишку. Під час цього процесу зберігаються протяжність і зв'язність вихідної області, проте більша частина вихідних пікселів переднього плану відкидається.

Скелет, який виходить після процесу скелетизації, виражає структурні зв'язки основних компонентів об'єкта і має розмір один піксель завширшки. Це дає змогу зберегти форму об'єкта й одночасно зменшити його розмір.

Отриманий скелет може бути використаний для подальшого аналізу зображення та виділення значущих ознак об'єкта[6].

Цей метод може бути корисним для розпізнавання рукописного тексту, але може страждати від неточностей під час роботи зі складними символами, що перекриваються.

### 3.2.7 Зображення в градаціях сірого (Grayscale)

Зображення в градаціях сірого - це варіант подання зображення, де кожен піксель кодує яскравість, а не колір. Це означає, що кожен піксель являє собою відтінок сірого, що варіюється від чорного до білого як на рис. 3.3. Переведення зображення в градації сірого спрощує обробку зображення, оскільки ми працюємо тільки з одним каналом, а не з трьома (червоний, зелений і синій). Друга причина полягає в тому, що багато методів попередньої обробки зображень, такі як бінаризація або порогова обробка, вимагають одноканальних зображень. Також зображення в градаціях сірого можуть бути більш ефективними під час розпізнавання тексту, оскільки колір зазвичай не відіграє великої ролі в розпізнаванні символів, а кольорові зображення можуть містити більше шуму.

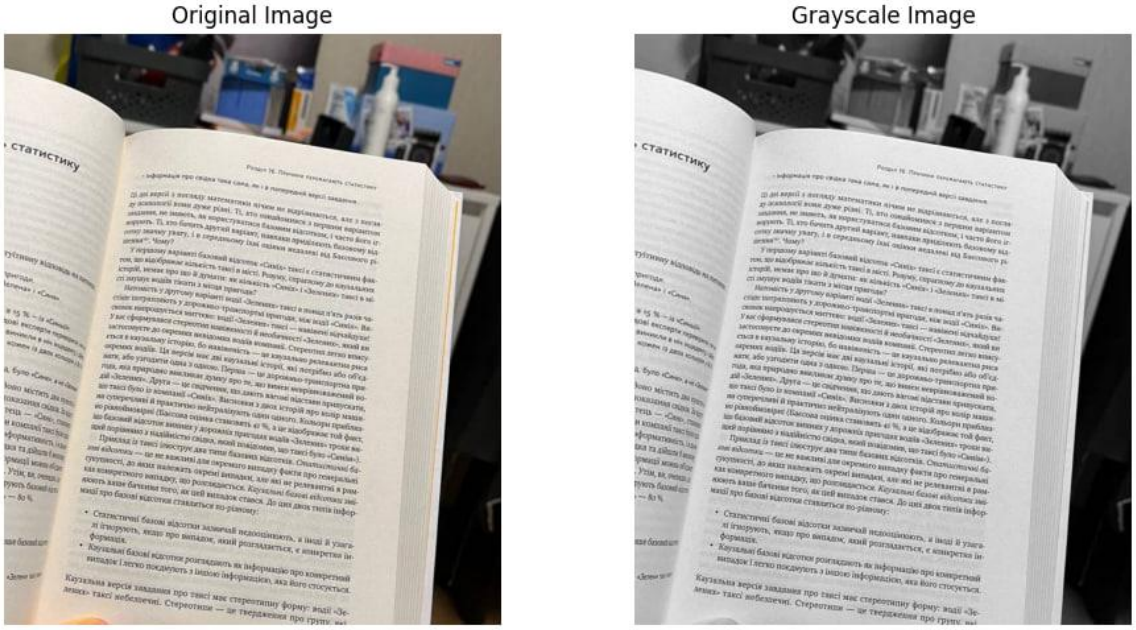



Рисунок 3.3. – Приклад зображення в градаціях сірого

### 3.2.7. Порогова обробка або бінаризація (Thresholding or Binarization)

Порогова обробка або бінаризація - це важливий етап попередньої обробки в OCR, який перетворює зображення в градаціях сірого на чорно-біле зображення. На цьому етапі кожному пікселю зображення присвоюється одне з двох значень: або чорний, або білий, залежно від того, чи перевищує його яскравість заданий пороговий рівень. Це допомагає знизити складність зображення і спрощує наступні етапи розпізнавання тексту, прибираючи зайві деталі та залишаючи тільки текст. Проте вибір правильного порогового значення може бути складним завданням, оскільки він варіюється залежно від умов освітлення і якості зображення. На щастя, є різні методи, такі як адаптивна порогова обробка та метод Оцу, які автоматично обчислюють оптимальне порогове значення на основі характеристик зображення. Приклад можно побачити на рис. 3.4.

Принцип методу Оцу:

$$\sigma_{\omega}^2(t) = \omega_1(t)\sigma_1^2(t) + \omega_2(t)\sigma_2^2(t) \qquad (3.6)$$

де ваги $\omega_i$ — це ймовірності двох класів, розділених порогом t

$\sigma_i^2$— дисперсія цих класів

Такий підхід гарантує, що порогове значення буде обрано так, щоб максимально відрізнялись розподіли інтенсивностей пікселів в чорному та білому класах.

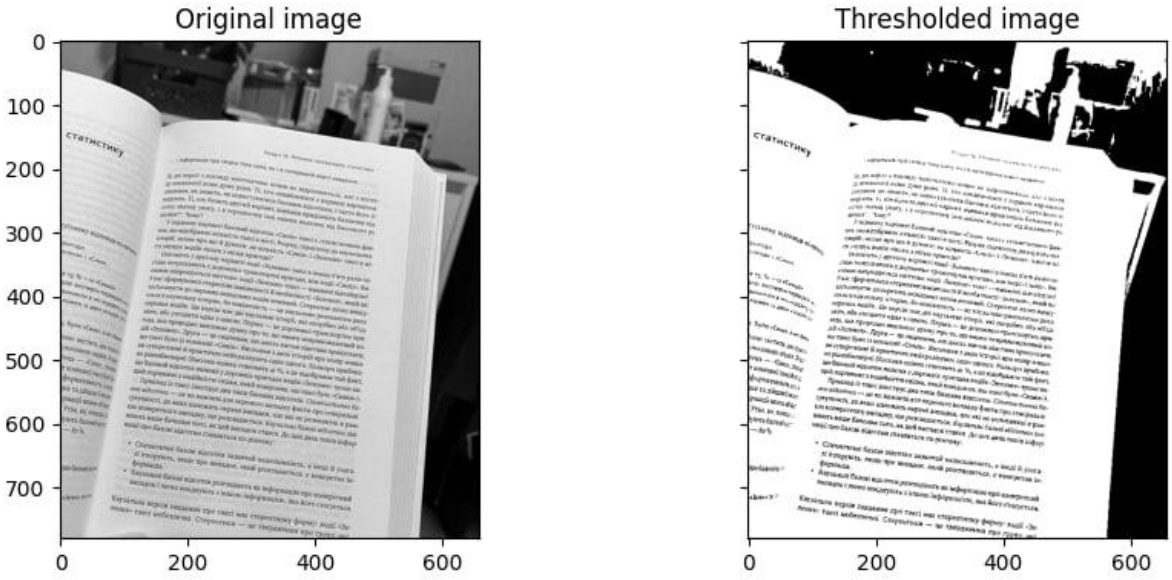



Рисунок 3.4. – Приклад порогового оброблення

## 3.3. Пошук та сегментація тексту

### 3.3.1. Розпізнавання на основі ручних ознак і людських знаннях

У цій роботі ми використовуватимемо підхід на основі ручних ознак, оскільки вже цього більш ніж достатньо для гарних результатів на не сильно зашумлених зображеннях друкованого тексту. Переваги цього підходу в тому, що нам не потрібні додаткові навчальні дані та моделі для навчання.

### 3.3.2. Використання шаблонів і матриць зіставлення

Методи, засновані на шаблонах і матрицях зіставлення, в основі мають якийсь "сканер", який порівнює шаблони із зображеннями, якщо певна

кількість умовних точок у матриці співпала, і це значення перевершує поріг, значить, цей фрагмент маркується як відповідний символ[8].

Ці збіги в матриці обчислюються за допомогою різних метрик схожості, наприклад, крос-кореляції, евклідова відстань або схожість на основі перетворення Хафа між шаблоном і вихідним зображенням.

Цей метод може досягати точних результатів, але тільки за ідеальних умов. Він не є універсальним, оскільки використовуються фіксовані шаблони, які не враховують різні шуми і фактори ззовні. Потрібна велика кількість шаблонів для покриття різноманітних шрифтів, однак це ніяк не захистить від можливих перешкод на зображенні.

### 1.3.3. Геометричні та статистичні ознаки

Геометричні та статистичні методи ґрунтуються на первинному вилученні ознак, а потім подальшому аналізі.

Геометричні методи описують форму і внутрішню структуру символів, водночас статистичні ґрунтуються на розподілах інтенсивності пікселів[7].

Після вилучення цих ознак можна використовувати алгоритми машинного навчання для подальшої класифікації.

Геометричні ознаки містять у собі:

- периметр (обчислюється довжина контуру символу)
- площа (кількість пікселів, що належать до символу)
- компактність (відношення площі символу до квадрата периметра, виділеного під нього)
- кут нахилу (кут відносно горизонтальної осі) тощо.

Статистичні ознаки містять у собі:

- медіана (інтенсивність пікселів)
- мода (найпоширеніше значення інтенсивності)
- середнє значення (середня інтенсивність пікселів усередині символу)

- стандартне відхилення (міра розкиду пікселів від середнього)
- дисперсія (середній квадрат відхилень від середнього значення) і тд [9].

Цей підхід має логічні й обґрунтовані причини, застосовні до багатьох стилів тексту. Він більш універсальний, за рахунок можливості інтерпретувати знання про один шрифт іншим. Однак він чутливий до шуму і різних деформацій із зовні.

### 1.3.4. Морфологічний аналіз

Морфологічний аналіз заснований на вивченні структури зображення. Він застосовує різні зміни і деформації до зображення, щоб за рахунок вибіркової втрати інформації, можна було витягти необхідні об'єкти.

Це сприяє швидкому аналізу, оскільки тут ми враховуємо мінімальну кількість елементів для аналізу, прибравши весь шум. Ця техніка використовується в основі системи людського зору[10].

Основні морфологічні операції:

- Ерозія

Ерозія зменшує розмір об'єктів на зображенні, видаляючи пікселі шляхом додавання, тим самим позбуваючись маленьких об'єктів і шумів.

Ерозія бінарного зображення A структурним елементом B визначається за формулою:

$$(A \ominus B) = \{z \in E \mid Bz \subseteq A\} \quad (3.7)$$

де

A - це бінарне зображення,

B - це структурний елемент,

Bz - це перенесення структурного елементу таким чином, що його центр співпадає з пікселем z(3.8)

$$Bz = \{b + z | b \in B\}, \forall_z \in E \quad (3.8)$$

Якщо піксель структурного елемента збігається з пікселем бінарного зображення, то виконується логічне додавання центрального пікселя структурного елемента з відповідним вихідного зображення.

- Нарощування (дилатація)

Нарощування збільшує розмір об'єктів на зображенні, об'єднуючи сусідні пікселі. Фактично це є зворотною операцією ерозії.

Формула дилатації:

$$(A \oplus B) = \bigcup_{b \in B} A_b \quad (3.9)$$

- Відкриття

Спершу виконується операція ерозії, а після застосовується операція нарощування, тим самим уникаючи сильного зменшення об'єктів на зображенні. Відкриття може бути використане для видалення шуму та збереження меж важливих об'єктів.

Формула відкриття:

$$(A \circ B) = ((A \ominus B) \oplus B) \quad (3.10)$$

- Закриття

Зворотний підхід до відкриття. Тобто спершу застосовується нарощування, а потім ерозія. Може використовуватися для видалення пробілів у символах і збереження розміру об'єктів[11].

Формула закриття:

$$(A \bullet B) = ((A \oplus B) \ominus B) \quad (3.11)$$

Морфологічний аналіз добре використовується на різних етапах процесу розпізнавання символів, включно з початковим обробленням

зображень, виокремленням текстових областей і сегментацією символів. Однак ефективність все ще залежить від рівня складності даних та їхньої зашумленості.

### 1.3.5. Структурний аналіз

Структурний аналіз передбачає розпізнавання на основі структури символів. Знаходяться і враховуються різні характеристики символів, такі як:

- Точки розгалуження
- Кінцеві точки скелета
- Замкнуті контури і тд.

У деяких більш ускладнених методах після вилучення структури, застосовують подальший аналіз графа суміжності для виділення слів. У таких графах вершини відповідають певним точкам символу (наприклад, кінцевим точкам), а ребра - з'єднанням між ними. Далі побудований граф порівнюється з еталонним. Якщо певний поріг збіжності подолано, то символ маркується так само, як і шаблон[12].

Перевага цього методу в тому, що він стійкий до деформацій і масштабування. Однак метод складний у реалізації та потребує врахування безлічі неочевидних моментів.

### 1.3.6. Зонування

Зонування - це процес поділу зображення на зони фіксованого розміру, а потім подальшого аналізу. Обчислюються характеристики кожної із зон, такі як дисперсія, контраст і щільність, а потім аналізуються моделями машинного навчання.

Розбиваючи зображення на більш дрібні фрагменти, зонування може поліпшити точність розпізнавання [13].

Основною перевагою алгоритму оптичного розпізнавання тексту, заснованого на зонуванні, є його можливість пояснення та інтерпретованість. Такий підхід може бути корисним під час виявлення помилок класифікації та пошуку способів оптимізації алгоритму.

Проте, алгоритм має і певні недоліки. По-перше, він може проявляти чутливість до зміни масштабу і обертання символів. У разі, якщо символи на зображенні представлені в різних масштабах або нахилені відносно один одного, це може спричинити помилки під час класифікації. Щоб усунути подібні проблеми, можна скористатися попередньою обробкою зображень, такою як масштабування і вирівнювання символів.

По-друге, цей метод може бути менш продуктивним щодо символів, які мають велику різноманітність написання, наприклад, рукописного тексту.

Існує й інша версія зонування - радіальна. Цей метод включає поділ зображення символу на концентричні кільця або сектори, що виходять із центру символу. Він має деякі переваги порівняно зі звичайною сіткою, оскільки він враховує радіальну структуру символів і може бути більш стійким до викривлень, що спричинені обертанням. Однак, він все ще чутливий до змін.

### 1.3.7. Проекційний профіль

Метод на основі проекційного профілю у своїй основі використовує гістограми інтенсивності пікселів для визначення меж текстових областей і символів.

Для початку обчислюються вертикальні та горизонтальні гістограми інтенсивності пікселів. Потім встановлюються порогові значення, які визначать, належить ця область до символів чи ні.

За допомогою цього методу можна як і сегментувати зображення, знаходячи область із текстом, так і визначати наявність символів у фіксованих областях.

Наприклад, розглянемо випадок із розпізнаванням наявності елементів у фіксованих областях.

У цьому прикладі ключове завдання розпізнати обрану відповідь у сітці відповідей.

Обчислюється горизонтальний і вертикальний проекційний профіль зображення як на рис. 3.5:

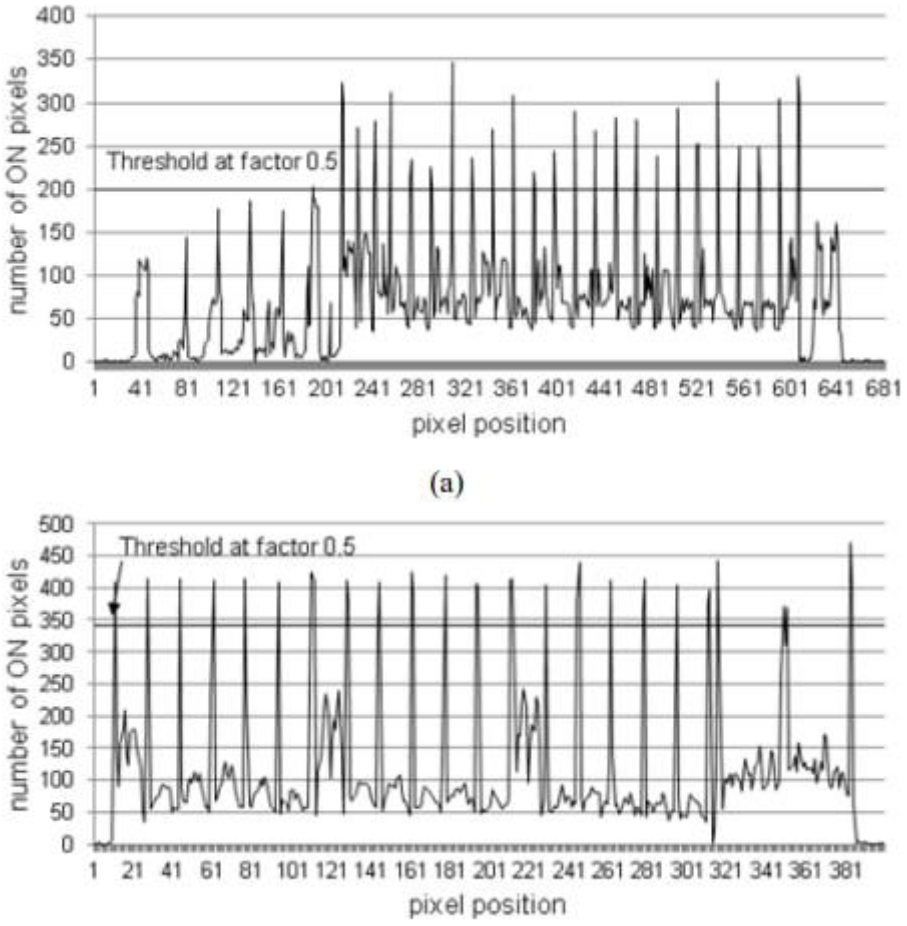



Рисунок 3.5. – Проекційний профіль бланку відповідей (а) горизонтальна проекція (б) вертикальна проекція. [14]

Ці гістограми дають змогу виявити сітку з відповідями. На графіках видно, що сітка рівномірно розподілена.

Далі застосовується порогова обробка, за допомогою якої виявляються всі значущі лінії.

Область, за якою ми хочемо стежити, визначається координатами як на рис. 3.6:

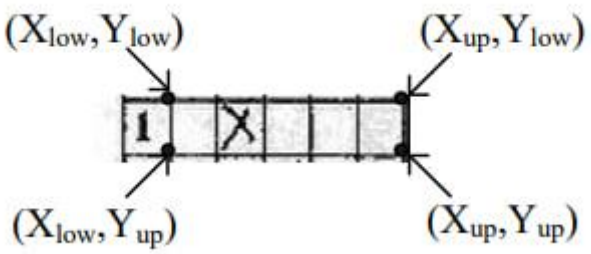



Рисунок 3.6. – Зона запитань представлена чотирикутником (Xup, Xlow, Yup, Ylow) [14]

Далі алгоритм робить аналогічні пункти, але для локальних фрагментів для кожної виділеної області та використовує порогову обробку, щоб визначити обрану відповідь (рис. 3.7).

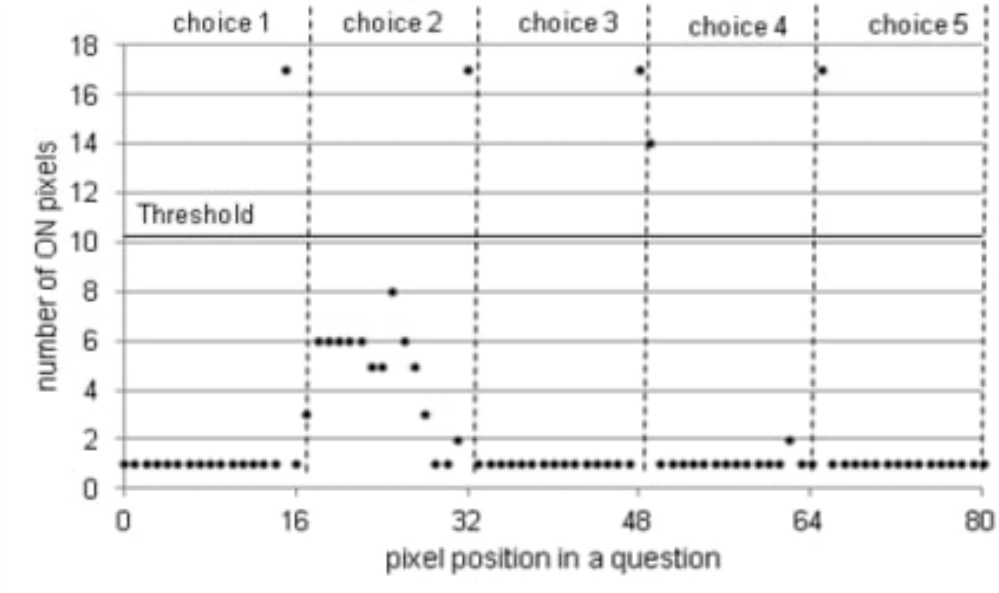



Рисунок 3.7. – Профіль проекції зони відповіді на питання [14]

Проекційний профіль успішно справляється з добре структурованими даними, чіткими межами і мінімальним шумом.

Основною перевагою цього методу є його простота в реалізації і не затратність на процесі. Однак, він схильний до помилок, якщо текст написаний нерівно, нахилений, якось деформований або перекреслений[15].

## 3.3. Розпізнавання тексту

### 3.3.1. Модель CNN

Згорткові нейронні мережі (Convolutional Neural Networks, CNN) є одним з основних типів глибоких нейронних мереж, які використовуються в обробці зображень. Суть "згортки" полягає в тому, що вона дає змогу витягувати та комбінувати окремі характеристики для досягнення найбільш точного результату. Це стає можливим завдяки здатності мережі скорочувати свої розміри на більш глубоких рівнях, що допомагає виділяти опорні ознаки (наприклад, контури та краї об'єкта). Що більше мережа "згортається" і заглиблюється в аналіз, то чіткіше форми конкретної частини об'єкта фіксуються на шарі.

CNN працює за принципом людського мозку в той момент, коли ми бачимо зображення. Подібно до того, як кожен нейрон реагує на стимули тільки в обмеженій ділянці візуального поля, що називається рецептивним полем у біологічній системі зору, кожен нейрон у CNN опрацьовує дані тільки у своєму рецептивному полі.

Згортання мережі відбувається завдяки ядру, що являє собою матрицю невеликого розміру. Це ядро ковзає по всім даним попереднього шару, послідовно обробляючи невеликі фрагменти інформації. Кожна така "обробка" є окремим міні-шаром, який на вході має рівно ту кількість елементів, яка є в ядрі, а на виході лише одне значення. Кожен із цих міні-шарів відповідає за свою унікальну характеристику. Потім ці міні-шари

об'єднуються в один шар, який менший за розміром, ніж вихідний. Таке згортання можна повторювати кілька разів, поки не буде досягнуто бажаного результату.

Структура CNN зазвичай складається з трьох основних типів шарів: згортувальних шарів, шарів пулінгу і повнозв'язних шарів (рис. 3.8).

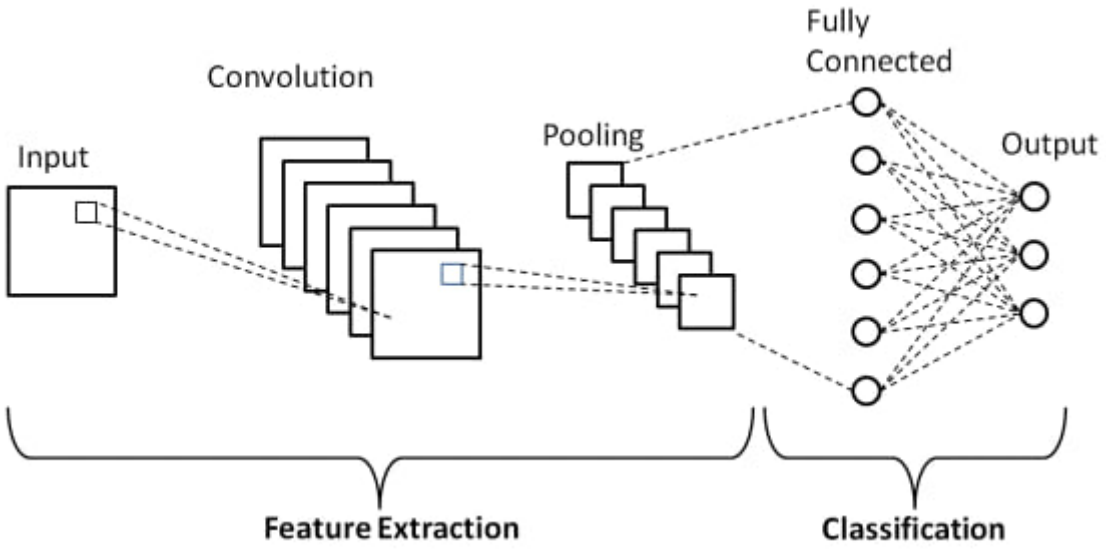



Рисунок 3.8. – Структура CNN [17]

1. Згортковий шар (Convolution layer, CONV)

Згортковий шар - це основний будівельний блок CNN. Він несе основне обчислювальне навантаження мережі. Цей шар використовує набір фільтрів, що навчаються, кожен з яких проходить по всьому зображенню, скануючи невелику область за раз (рис. 3.9). Кожен фільтр виділяє різні ознаки зображення, наприклад, межі, кольори або текстури.

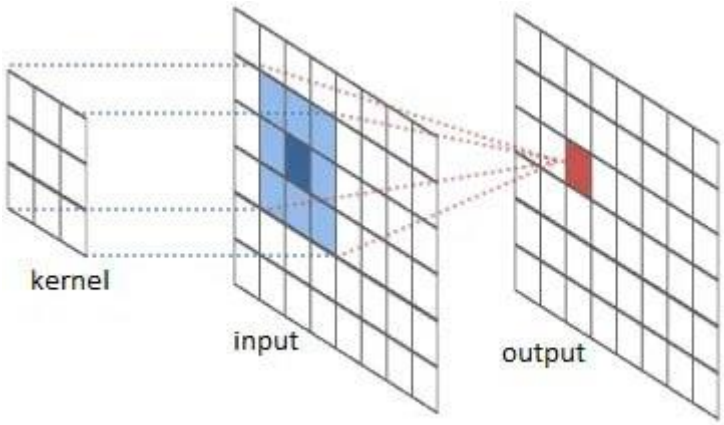



Рисунок 3.9. – Принцип роботи згорткового шару [18]

Шар згортки використовує фільтри, які виконують операції згортки під час сканування входу I відносно його розмірів. Гіперпараметри включають розмір фільтра F і крок S. Результуючий вихід O називається картою особливостей або картою активації [16].

2. Шар пулінгу

Цей шар скорочує просторові розміри зображення (ширину і висоту), зберігаючи при цьому найважливіші ознаки (рис. 3.10). Це допомагає зменшити обсяг обчислень і контролювати перенавчання.

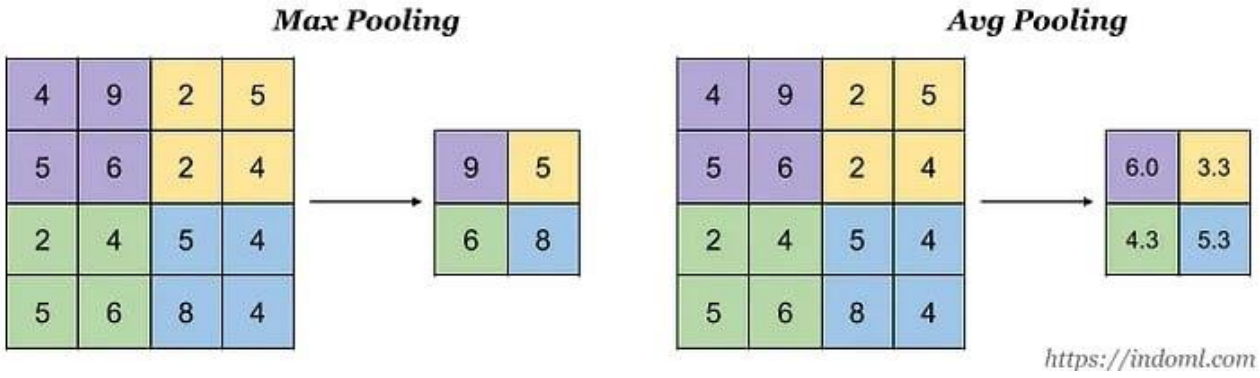



Рисунок 3.10. – Приклад максимального і середнього пулінгу [18]

Ця операція зазвичай застосовується після шару згортки, що забезпечує певну просторову інваріантність.

3. Повнозв'язний шар

Наприкінці CNN, після кількох згорткових і пулінгових шарів, ознаки зображення перетворюються на одновимірний вектор і подаються на один або кілька повнозв'язних шарів для класифікації або регресії. Нейрони в цьому шарі повністю пов'язані з усіма нейронами попереднього і наступного шару. Далі до них застосовується якась нелінійна функція (на зразок сигмоїди, гіперболічного тангенса або ReLU).

### 3.3.2. Гіперпараметри CNN

1) Filter

Зазвичай, шари, які ближче до вхідних даних, мають меншу кількість фільтрів, в порівнянні з більш глибокими шарами, які можуть мати значно більше фільтрів. Але природно це відбувається якраз навпаки, у міру поглиблення мережі, кількість фільтрів скорочується, оскільки втрачається частина ознак. Гіперпараметр Filter якраз дає змогу зберігати баланс обчислень у кожному шарі.

2) Stride

Для операції згортки або об'єднання крок (Stride) позначає кількість пікселів, на яку переміщується вікно після кожної операції.

3) Padding (або Zero-padding)

Padding позначає процес додавання нулів P по обидва боки від меж вхідних даних. Це значення можна вказати вручну або встановити автоматично за допомогою одного з трьох режимів[023]:

- Valid

Не додає нулі зовсім:

$$\mathrm{P} = 0 \tag{3.12}$$

- Same

До вхідного зображення додається достатньо нулів, щоб вихідне зображення мало той же розмір, що і вхідне зображення, при умові, що крок дорівнює одиниці:

$$\mathrm{P}start = \frac{[\mathbf{S}\lceil\frac{I}{S}\rceil - \mathbf{I} + \mathbf{F} - \mathbf{S}]}{2} \tag{3.13}$$

$$\mathbf{P}end = \frac{[\mathbf{S}\lceil\frac{I}{S}\rceil - \mathbf{I} + \mathbf{F} - \mathbf{S}]}{2} \tag{3.14}$$

де I – розмір вхідного зображення

F – розмір фільтра

S – крок

- Full

Full padding максимально збільшує розмір вихідного зображення. Кількість пікселів, які додаються до вхідного зображення, залежить від розміру фільтра. Якщо розмір фільтра дорівнює F, то до вхідного зображення додаються F-1 рядків і стовпців(3.16):

$$\mathbf{P}start \in [[\mathbf{0}, \mathbf{F} - \mathbf{1}]] \tag{3.15}$$

$$\mathbf{P}end = \mathbf{F} - \mathbf{1} \tag{3.16}$$

### 3.3.3. Підбір гіпперпараметрів

Метою оптимізації гіперпараметрів у машинному навчанні є пошук оптимальних гіперпараметрів певного алгоритму, які забезпечують найкращу

продуктивність, виміряну на валідаційному наборі даних. Нижче розглянемо формулу випадкового пошуку.

Оптимізація гіперпараметрів представлена у вигляді рівняння як:

$$x^* = \arg\min_{x \in X} f(x) \tag{3.17}$$

Де f(x) - оцінка, яку потрібно мінімізувати, наприклад, середньоквадратичне відхилення або частоту помилок на валідаційному наборі;

x* - набір гіперпараметрів, який дає найнижче значення оцінки

x - будь-яке значення в області X[20]

Проблема з оптимізацією гіперпараметрів полягає в тому, що оцінка цільової функції для знаходження оцінки є надзвичайно дорогою, потому что необхідно пройтися за всіма параметрами по черзі.

Можна використовувати більш спрощені та зручні методи. Одним із таких підходів скористаємося і ми.

Bayesian Optimization

Байєсівські підходи, на відміну від випадкового пошуку, відстежують минулі результати оцінювання, які вони використовують для формування ймовірнісної моделі, що відображає гіперпараметри на ймовірність оцінки за цільовою функцією:

$$P(score|hyperparameters) \tag{3.18}$$

Методи байєсівської оптимізації є ефективними, оскільки вони обирають наступні гіперпараметри обґрунтовано. Вони базуються на принципі приділення додаткового часу для покращенного вибору наступних гіперпараметрів, що в свою чергу дозволяє зменшити кількість викликів цільової функції. Оцінюючи гіперпараметри, які здаються більш перспективними на основі попередніх результатів, байєсівські методи можуть знайти кращі налаштування моделі, ніж випадковий пошук, за меншу кількість ітерацій.

В контексті практичної реалізації, час, витрачений на визначення наступних гіперпараметрів, є мінімальним у порівнянні з часом, потрібним для обчислення цільової функції.

### 3.3.4. Найбільш популярні функції активації в CNN

Функції активації використовуються після кожного шару згортки для додавання нелінійності до моделі.

1) ReLU та модифікації її

ReLU (Rectified Linear Unit) обнуляє всі негативні входи, а позитивні входи залишаються без змін за формулою:

$$g(z) = \max(0, z) \tag{3.19}$$

Графік ReLU зображено на рис.3.11.

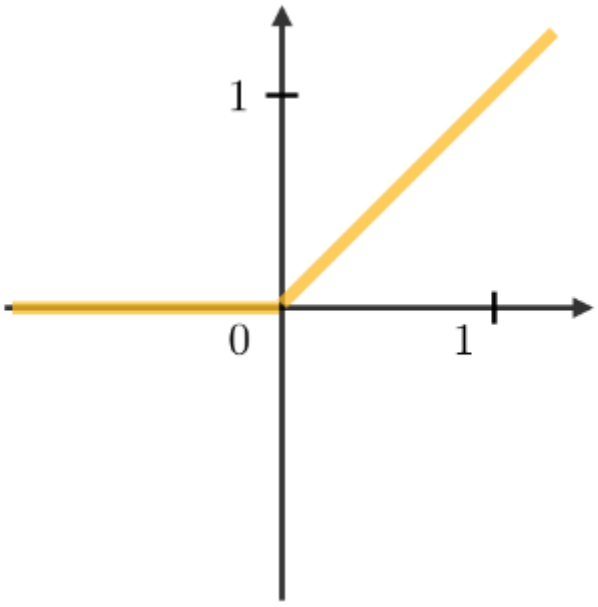



Рисунок 3.11. – Графік ReLU [16]

Leaky ReLU, Parametric ReLU, ELU, Swish – це варіації звичайної ReLU, які намагаються усунути проблему "вмираючого ReLU" , коли нейрони стають неактивними і більше не вчаться.

2) Softmax

Softmax використовується на вихідних шарах нейронних мереж для багатокласової класифікації, оскільки вона може перетворити набір вхідних значень у ймовірності для кожного класу за формулою:

$$p_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{3.20}$$

де $p_i$ – це один з елементів вихідного вектора

### 3.3.5. Удосконалені архітектури CNN

С розвитком глибокого навчання, багато моделей і архітектур було запропоновано для різних типів завдань. Зокрема, для завдань обробки зображень, де просторовий контекст і ієрархія особливостей мають важливе

значення. CNN моделі мають ряд ключових властивостей, що роблять їх ефективними для обробки зображень, включаючи здатність визначати локальні особливості через конволюцію, зменшення просторового розміру через пулінг і створення ієрархійних представлень через багатошарові структури. Далі буде розглянуто декілька удосконалених архітектур CNN, що заслуговують на увагу.

1. LeNet:

LeNet-5, запропонована Яном ЛеКуном і його командою в 1998 році, була однією з перших згорткових нейронних мереж і зробила величезний вплив на розвиток цього класу моделей. Її розробили спеціально для завдання розпізнавання рукописних цифр, і використали для автоматичного розпізнавання поштових індексів у США.

Структура LeNet-5 складається з 7 шарів, включно зі згортковими, пулінговими та повнозв'язними шарами. Ця модель демонструє ключові принципи роботи згорткових мереж: локальне сприйняття через згортку, стійкість до зрушень через підвибірку та ієрархічне[19]. Модель LeNet-5 зображено на рис. 3.12.

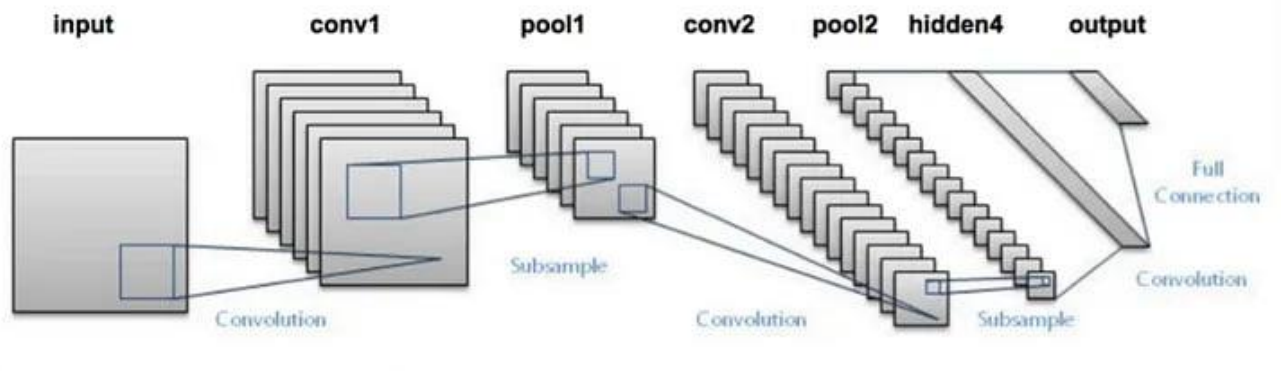



Рисунок 3.12. – LeNet-5 [19]

2. AlexNet:

AlexNet, запропонована Алексом Крижевським, Іллею Суцкевером і Джеффрі Гінтоном у 2012 році, була першою моделлю, що застосувала глибоку згорткову нейронну мережу для завдання класифікації зображень на

великому наборі даних (ImageNet). Вона значно перевершила всі попередні моделі і стала важливою віхою в розвитку глибокого навчання.

Структура AlexNet складається з 8 шарів, включно з 5 згорткових і 3 повнозв'язних шари. На відміну від LeNet, AlexNet використовує глибшу архітектуру і вводить кілька нових технік, як-от використання функції активації ReLU, техніки проріджування (dropout) і навчання з використанням графічних процесорів (GPU).

У контексті OCR, ці класичні архітектури можуть бути використані як відправна точка для створення більш складних моделей. Зокрема, вони можуть бути застосовані для виявлення і розпізнавання тексту на зображеннях. Проте сучасні моделі, як-от ResNet або DenseNet, зазвичай перевершують їх за ефективністю завдяки глибшим архітектурам і застосуванню додаткових технік, як-от залишкове навчання. Модель AlexNet зображено на рис. 3.13.

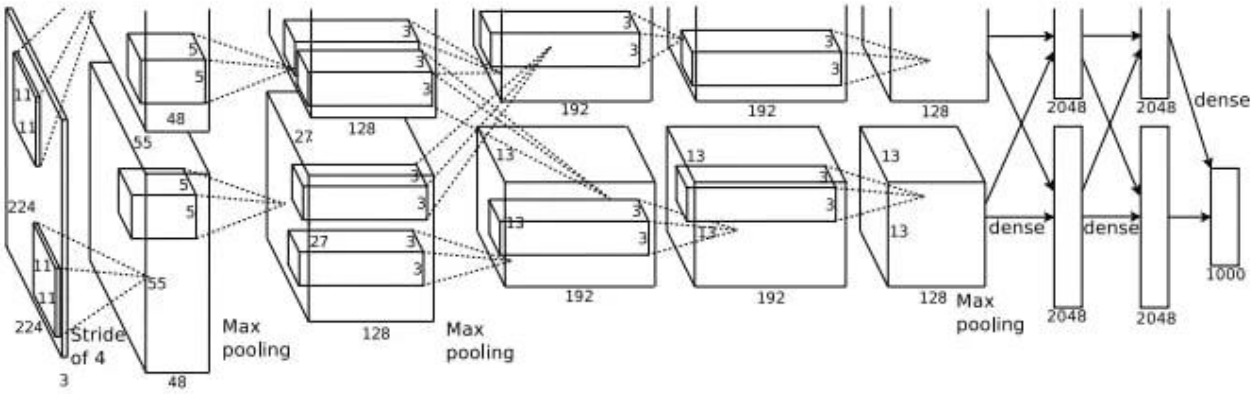



Рисунок 3.13. – AlexNet [19]

3. VGGNet:

VGGNet, запропонована групою дослідників з Оксфордського університету 2014 року, являє собою ще одну знакову архітектуру в галузі згорткових нейронних мереж. Відмінною рисою VGGNet є її глибина: вона складається з 16-19 шарів, включно зі згортковими і повнозв'язними шарами.

Крім того, VGGNet використовує маленькі згортки розміром 3x3 у всіх згорткових шарах, що дає їй змогу ефективно витягувати складні ознаки. Модель VGGNet зображено на рис. 3.14.

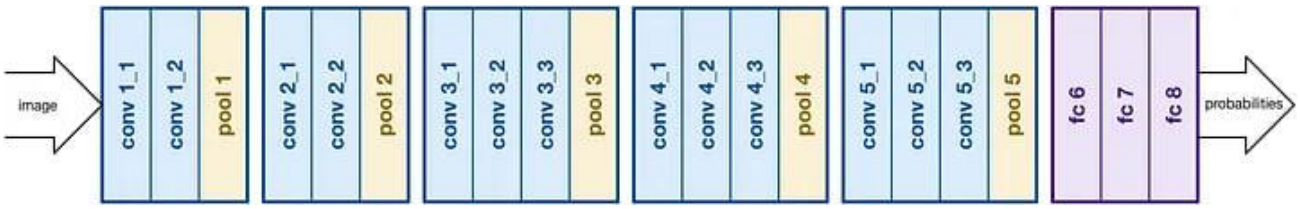



Рисунок 3.14. – VGGNet [19]

4. ResNet:

ResNet (Residual Network), запропонована групою дослідників із Microsoft Research 2015 року, є ще однією знаковою моделлю в галузі глибокого навчання. Відмінною рисою ResNet є її використання залишкових блоків, які дозволяють моделі ефективно навчатися, навіть коли вона має дуже велику глибину (понад 100 шарів). Це досягається шляхом додавання "залишкових з'єднань", які дозволяють сигналу "пропускати" один або кілька шарів. Модель ResNet зображено на рис. 3.15.

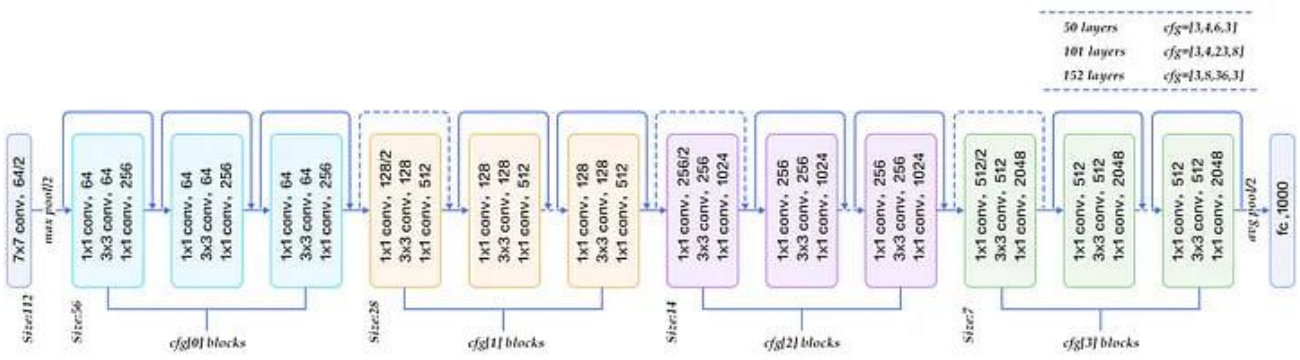



Рисунок 3.15. – ResNet [19]

5. DenseNet:

DenseNet (Densely Connected Convolutional Network), запропонована групою дослідників у 2016 році, є моделлю, яка внесла нову ідею в архітектуру згорткових нейронних мереж. У DenseNet кожен шар з'єднаний з

усіма наступними шарами, утворюючи щільну мережу з'єднань. Це дає змогу DenseNet ефективно перевикористовувати ознаки і знижує кількість необхідних параметрів. Модель DenseNet зображено на рис. 3.16.

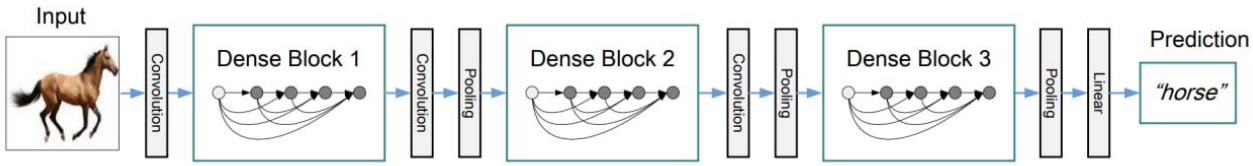



**Figure 2:** A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

Рисунок 3.16. – DenseNet [19]

6. EfficientNet

EfficientNet, представлений 2019 року, є новітнім удосконаленням у сфері згорткових нейронних мереж. Основна ідея EfficientNet полягає в балансуванні між глибиною, шириною і роздільною здатністю мережі, що дає змогу досягти високої ефективності з меншою кількістю параметрів. EfficientNet досягає чудових результатів у задачах комп'ютерного зору і може бути застосований у задачах оптичного розпізнавання тексту для отримання більш точних результатів. Модель EfficientNet зображено на рис. 3.17.

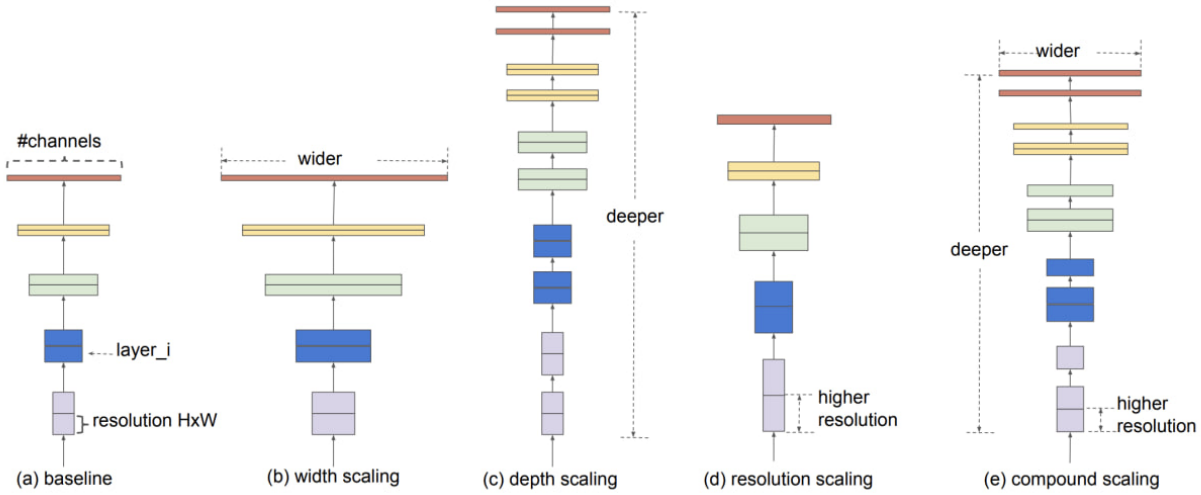



*Figure 2.* **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

Рисунок 3.17. – EfficientNet [19]

Усі ці моделі можуть бути застосовані в контексті оптичного розпізнавання тексту, хоча деякі з них можуть потребувати адаптації для роботи з послідовностями символів.

# ВИСНОВОК ДО РОЗДІЛУ 3

У цьому розділі були розглянуті ключові бібліотеки, з якими ми будемо працювати далі, таки як Python, TensorFlow, Matplotlib, OpenCV, які дозволили розробити ефективну та продуктивну систему OCR. Платформа Google Colab сприяла гнучкості розробки, надаючи зручне середовище для виконання великого обсягу обчислень з даними. Aiogram був вибраний для розробки Telegram бота через його широкі можливості та гнучкість.

Було підкреслено важливість попередньої обробки даних R, яка включає в себе процеси нормалізації, вирівнювання, масштабування зображення, видалення шуму, скелетизації, перетворення зображення в градації сірого, та порогової обробки або бінаризації.

Так само було розглянуто методи пошуку та сегментації тексту та розглянута детально модель CNN, її гіперпараметри, способи поліпшення їх і вдосконалені моделі CNN.

# РОЗДІЛ 4 ЕКСПЕРИМЕНТИ ТА РЕЗУЛЬТАТИ

Під час розроблення системи було проведено купу експериментів для підбору найоптимальнішої системи в рамках підходу посимвольного розпізнавання.

## 4.1. Початкове попереднє оброблення зображень

Вихідне зображення перед початком сегментування необхідно підготувати. У цій роботі виконується такими функціями:

- load_image(): метод завантажує зображення з вказаного шляху, перетворюючи його в масив пікселів

- to_gray(): перетворює зображення в градації сірого, використовуючи лише одне значення яскравості замість трьох кольорових каналів

- contrast(): покращує контрастність зображення, нормалізуючи всі значення пікселів до діапазону 0-255

- as_white_background(): переконується, що фон зображення є білим. Якщо середнє значення пікселів менше 128, зображення інвертується, так що текст стає темним, а фон – світлим

## 4.2. Модель сегментації та пошуку тексту

Сегментування тексту в даній системі виконується за допомогою використання геометричних та статистичних ознак, морфологічного аналізу, а також на основі профільної проекції.

Одним з ключових аспектів нашого підходу є використання геометричних та статистичних ознак разом з проекційним профілем. Ми аналізуємо гістограми зображення, щоб виявляти горизонтальні та вертикальні лінії, як це робиться в методах get_lines_positions та get_words_positions, які представлені на рис.4.1. Застосування цих методів дозволяє нам враховувати розташування та розміщення літер і слів на зображенні, відображаючи тим самим геометричні особливості тексту.

```python
def get_lines_positions(self, im, threshold=.002):
    hist = self.get_histogram(self.get_edges(im))
    threshold = hist.min() + threshold * (hist.max() - hist.min())

    height = hist.shape[0]
    lines = []

    pos = 0
    while True:
        while pos < height and hist[pos] < threshold: pos += 1
        if pos == height:
            break

        line_start = pos
        while pos < height and hist[pos] >= threshold: pos += 1
        line_end = pos
        lines.append((line_start, line_end))
        if pos == height:
            break

    lines = [i for i in lines if abs(i[0] - i[1]) > 5]
    return lines
```

```python
def get_words_positions(self,line, threshold=0.1, space_threshold=0.5):
    if len(line.shape) > 2:
        line = cv2.cvtColor(line, cv2.COLOR_BGR2GRAY)

    hist = self.get_histogram(line, horiz=True)
    threshold = hist.max() - threshold * (hist.max() - hist.min())

    h_hist = self.get_histogram(line)
    letters_height = h_hist / h_hist.mean() / 2
    letters_height = 1 - np.round(letters_height)
    letters_height = letters_height.sum()
    space_threshold = int(round(letters_height * space_threshold))
    width = hist.shape[0]
    words = []

    pos = 0
    while True:
        while pos < width and hist[pos] >= threshold: pos += 1
        if pos == width:
            break

        word_start = pos
        while pos < width and hist[pos] < threshold: pos += 1
        word_end = pos
        new_word = (word_start, word_end)
        if len(words) != 0:
            prev_words = words[-1]
            if word_start - prev_words[1] < space_threshold:
                words[-1] = (words[-1][0], word_end)
            else:
                words.append(new_word)
        else:
            words.append(new_word)

        if pos == width:
            break

    words = [i for i in words if abs(i[0] - i[1]) > 5]
    return words

def crop_word(self, img, word):
    start,end = word
    cropped_word = img[:, start:end]

    return cropped_word
```

Рисунок 4.1. – Код функцій get_lines_positions та get_words_positions

Метод get_letters_positions (рис. 4.2) використовує геометричний аналіз для визначення місцезнаходження окремих літер в межах слів. Він використовує функції бібліотеки OpenCV, такі як cv2.dilate та cv2.findContours, для виявлення та виділення окремих літер, що свідчить про нашу залежність від геометричних аспектів тексту.

```python
def get_letters_positions(self, word_image):
    gray = cv2.cvtColor(word_image, cv2.COLOR_BGR2GRAY)
    _, threshold = cv2.threshold(gray, 90, 255, cv2.THRESH_BINARY_INV)

    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (2,2))
    dilated = cv2.dilate(threshold, kernel, iterations=1)

    labels = measure.label(dilated, connectivity=2, background=0)
    positions = []

    for label in np.unique(labels):
        if label == 0:
            continue

        mask = np.zeros(dilated.shape, dtype="uint8")
        mask[labels == label] = 255

        contours, _ = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

        for contour in contours:
            x, y, w, h = cv2.boundingRect(contour)
            if w > 2 and h > 10:
                positions.append((x, y, w, h))

    return positions
```

Рисунок 4.2. – Код функції get_letters_positions

Морфологічний аналіз використовується в нашому підході через використання фільтра Canny в методі get_edges, що визначає контури на зображенні. Це дуже важливий крок у виявленні текстових ліній на зображенні, що дозволяє нам точніше розподілити текст на різні сегменти для подальшого аналізу.

Ключове зображення, яке використовувалося для тестування всіх етапів зображено на рис.4.3:

Мета видалення шуму - мінімізувати або повністю усунути перешкоди для поліпшення якості вилучення тексту. Шум на зображенні може набувати різних форм: від випадкових пікселів різного кольору до складніших артефактів, таких як нерівності або плями на фотографії. Шум може бути спричинений різними факторами, включно з умовами зйомки, якістю камери або процесом сканування документів. Видалення може бути досягнуто за допомогою різних технік і алгоритмів, включно з морфологічними операціями (наприклад, ерозією, розширенням, відкриттям і закриттям), згладжуванням (наприклад, розмиттям Гауса або медіанним фільтром) та іншими спеціалізованими методами, такими як усунення шуму на основі статистичних методів, які ми розглядали раніше.

Рисунок 4.3. – основне тестове зображення

Результати сегментації зображені на рис. 4.4:

Рисунок 4.4. – Приклад результатів сегментації (перші 70 символів)

Довелося протестувати різні варіанти гіпперпараметрів, щоб досягти гарного результату на основі цього підходу.

Як бачимо, і без глибокого навчання модель непогано так виділяє символи.

## 4.3. Модель розпізнавання тексту

У межах цієї системи буде реалізовано архітектуру моделі CNN для послідовного виявлення символів.

На вхід подаватимуться сегментовані зображення, попередньо оброблені для поліпшення якості мережі.

На виході буде видаватися клас літери, який вважається найбільш ймовірним за даними мережі.

### 4.3.1. Попереднє оброблення сегментованих зображень

Код функції попередньої обробки сегментованих зображень зображен на рис.4.5:

```python
def crop_img(self,img):
    if img is None or img.size == 0:
        raise ValueError("Invalid input image")

    coords = np.column_stack(np.where(img > 0))

    if coords.size == 0:
        raise ValueError("No non-zero pixels found")

    # обчислюємо обмежувальний прямокутник для ненульових координат і обрізаємо зображення
    x_min, y_min = coords.min(axis=0)
    x_max, y_max = coords.max(axis=0)
    cropped_img = img[x_min:x_max+1, y_min:y_max+1]

    if cropped_img.size == 0:
        raise ValueError("Cropped image is empty")

    # Обчислюємо нові розміри зображення, зберігаючи співвідношення сторін
    height, width = cropped_img.shape[:2]
    if height > width:
        new_height = 28
        new_width = int(width * new_height / height)
    else:
        new_width = 28
        new_height = int(height * new_width / width)

    resized_img = cv2.resize(cropped_img, (new_width, new_height))

    if resized_img.size == 0:
        raise ValueError("Resized image is empty")

    # ширина і висота білих смуг, які потрібно додати
    padding_height = 28 - new_height
    padding_width = 28 - new_width

    if padding_height < 0 or padding_width < 0:
        raise ValueError("Padding is negative. Check the resized image dimensions.")

    top_border_height = padding_height // 2
    bottom_border_height = padding_height - top_border_height

    left_border_width = padding_width // 2
    right_border_width = padding_width - left_border_width

    padded_img = cv2.copyMakeBorder(resized_img, top_border_height, bottom_border_height,
                                    left_border_width, right_border_width, cv2.BORDER_CONSTANT, value=255)
    padded_img = cv2.bitwise_not(padded_img)

    return padded_img
```

Рисунок 4.5. – Код функції попередньої обробки сегментованих зображень

Цей код реалізує метод crop_img, що використовується для обрізки і масштабування зображень. Початково він перевіряє, чи валідне вхідне зображення, знаходить координати ненульових пікселів і обчислює обмежувальний прямокутник для них. Зображення обрізається по цьому прямокутнику. Потім обчислюються нові розміри зображення, зберігаючи

при цьому співвідношення сторін, і виконується зміна розміру зображення. Код також обробляє можливі помилки, пов'язані з некоректними вхідними даними або некоректно обрізаними зображеннями. Нарешті, зображення доповнюється білими смугами, якщо воно менше заданого розміру, і виконується інверсія кольорів для отримання зображення з білими символами на чорному фоні. Це можно побачити на рис. 4.6.

Рисунок 4.6. – Приклад результатів сегментації після обробки

### 4.3.2. Експерименти

1) Архітектура першої моделі (рис. 4.7):

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=(28,28,1)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(y_train.shape[1], activation='softmax'))
```

Рисунок 4.7. – Архітектура першої моделі

Ця модель є послідовною нейронною мережею, що складається з декількох шарів. Першими двома шарами є шари згортки (Conv2D) з активаційною функцією ReLU та розміром ядра 3x3. Перший з них приймає

вхідні дані розміром 28x28x1. Далі йде шар максимального пулінгу (MaxPooling2D) з розміром пулу 2x2 для зменшення розмірності даних. Шар Dropout з коефіцієнтом 0.25 застосовується для запобігання перенавчання, відкидаючи деякі нейрони. Після цього, дані вирівнюються за допомогою шару Flatten. Ще один Dense шар з 128 нейронами та активацією ReLU додається до моделі, після якого застосовується ще один Dropout шар з коефіцієнтом 0.5. В кінці моделі розташований Dense шар, який відповідає кількості класів у навчальному наборі даних, і використовує функцію активації softmax для вибору найбільш ймовірного класу.

Досить проста модель, яка видавала слабкі результати.

2) Покращена архітектура

За допомогою різних експериментів вдалося визначитися з архітектурою, яка видавала найбільш вдалі результати(рис. 4.8):

```
def _build_model(self):
    model = Sequential()

    model.add(Conv2D(input_shape=self.input_shape, activation='relu', filters=32, kernel_size=3, padding='same'))
    model.add(Conv2D(activation='relu', filters=32,kernel_size=3, padding='same'))
    model.add(Conv2D( activation='relu', filters=32,kernel_size=5,  padding='same'))
    model.add(Dropout(0.4))

    model.add(Conv2D(activation='relu', filters=64,kernel_size=3,padding='same'))
    model.add(Conv2D(activation='relu', filters=64, kernel_size=5,padding='same'))

    model.add(Conv2D(activation='relu',filters=64,kernel_size=7, padding='same'))

    model.add(Dropout(0.4))

    model.add(Flatten())
    model.add(Dense(units=128, activation='relu'))
    model.add(Dropout(0.4))

    model.add(Dense(units=self.num_classes, activation='softmax'))
```

Рисунок 4.8. – Архітектура поліпшеною моделі

Навчання моделі було виконано на 19 епохах, екстрено завершившись за допомогою методу ранньої зупинки (EarlyStopping). Результати навчання можна побачити на рис.4.9.

```
Epoch 1/30
63/63 [==============================] - 11s 69ms/step - loss: 3.7437 - accuracy: 0.0992 - val_loss: 1.6723 - val_accuracy: 0.5346
Epoch 2/30
63/63 [==============================] - 3s 51ms/step - loss: 1.7083 - accuracy: 0.4813 - val_loss: 0.6404 - val_accuracy: 0.7472
Epoch 3/30
63/63 [==============================] - 3s 51ms/step - loss: 1.0573 - accuracy: 0.6274 - val_loss: 0.5275 - val_accuracy: 0.7693
Epoch 4/30
63/63 [==============================] - 4s 61ms/step - loss: 0.8547 - accuracy: 0.6828 - val_loss: 0.4577 - val_accuracy: 0.7889
Epoch 5/30
63/63 [==============================] - 3s 51ms/step - loss: 0.7299 - accuracy: 0.7209 - val_loss: 0.4112 - val_accuracy: 0.8200
Epoch 6/30
63/63 [==============================] - 3s 51ms/step - loss: 0.6253 - accuracy: 0.7509 - val_loss: 0.3895 - val_accuracy: 0.8215
Epoch 7/30
63/63 [==============================] - 4s 64ms/step - loss: 0.5912 - accuracy: 0.7682 - val_loss: 0.3471 - val_accuracy: 0.8455
Epoch 8/30
63/63 [==============================] - 3s 51ms/step - loss: 0.5088 - accuracy: 0.8013 - val_loss: 0.4455 - val_accuracy: 0.7939
Epoch 9/30
63/63 [==============================] - 3s 51ms/step - loss: 0.4614 - accuracy: 0.8204 - val_loss: 0.3267 - val_accuracy: 0.8546
Epoch 10/30
63/63 [==============================] - 4s 65ms/step - loss: 0.4350 - accuracy: 0.8313 - val_loss: 0.4341 - val_accuracy: 0.8134
Epoch 11/30
63/63 [==============================] - 3s 51ms/step - loss: 0.3880 - accuracy: 0.8492 - val_loss: 0.3692 - val_accuracy: 0.8475
Epoch 12/30
63/63 [==============================] - 4s 66ms/step - loss: 0.3929 - accuracy: 0.8509 - val_loss: 0.3105 - val_accuracy: 0.8746
Epoch 13/30
63/63 [==============================] - 4s 62ms/step - loss: 0.3816 - accuracy: 0.8510 - val_loss: 0.3243 - val_accuracy: 0.8676
Epoch 14/30
63/63 [==============================] - 3s 52ms/step - loss: 0.3464 - accuracy: 0.8659 - val_loss: 0.2913 - val_accuracy: 0.8776
Epoch 15/30
63/63 [==============================] - 3s 52ms/step - loss: 0.3584 - accuracy: 0.8648 - val_loss: 0.3324 - val_accuracy: 0.8661
Epoch 16/30
63/63 [==============================] - 4s 59ms/step - loss: 0.3336 - accuracy: 0.8707 - val_loss: 0.3417 - val_accuracy: 0.8606
Epoch 17/30
63/63 [==============================] - 4s 58ms/step - loss: 0.3098 - accuracy: 0.8835 - val_loss: 0.3531 - val_accuracy: 0.8541
Epoch 18/30
63/63 [==============================] - 3s 51ms/step - loss: 0.3150 - accuracy: 0.8779 - val_loss: 0.3401 - val_accuracy: 0.8626
Epoch 19/30
63/63 [==============================] - 3s 51ms/step - loss: 0.2965 - accuracy: 0.8860 - val_loss: 0.3523 - val_accuracy: 0.8641
```

Рисунок 4.9. – Результати навчання моделі

Після поєднання сегментації та розпізнавання можемо оцінити якість моделі на основі тестового зображення (рис. 4.3). Результати видно на рис. 4.10:

```
CNN модель(4) з більш слабкою аугментацією даних навчена на 19 епохах:

иетв видвленни шуиу  ифнфифзуввти бво повнфстю
усунути перешкоди дли полфпшенни икостф вилученни
тексту шуи нв ьзорвженнф иоже бнвуввти рфзних
фори вфд випвдкових пфксепфв рфзноф о кольору до
склвднфших фвртевк фві гвких ик нерфвнос бво плиии
нв фотоф фрвфї шуи иоже бу и спричинении рфзниии
фвк горвииі вкпючно з уиоввии зиоикиі икфстю
квиери бво процесои сквнуввнни докуиентфв
видвпенни иоже бути досигнуто зв допоиогою рфзних
гехнфк впф орич ифві вкпючно з фиоролоф фчниии
опервцфиии фнвприклвді ерозфєюі розширенниі
вфдкриттии фзвкриттиі зглвджуввннии фнвприклвді
розиич ии вусв бво иедфвннии ффльч фрои гв фншиии
спецфвпфзоввниии иетодвиіі твкиии ик усуненни
шуиу нв основф ствтистичних иетодфві икф ии
розгпидвли рвнфше
```

Рисунок 4.10. – Результати роботи поліпшеної архітектури CNN

3) Оптимізація гіпперпараметрів

Далі за допомогою Keras Tuner можна автоматично підібрати гіперпараметри. Keras Tuner - це бібліотека, яка допомагає вам підібрати оптимальний набір гіперпараметрів для моделі.

В даному проекті Bayesian Optimization використовується для ефективного підбору гіперпараметрів моделі. Нижче можна побачити код на рис. 4.11 та результати після виконнаня на рис.4.12.

```
def build_model(hp):
    model = Sequential()

    model.add(Conv2D(input_shape=input_shape,
                     activation='relu',
                     filters=hp.Int('conv_1_filters', min_value=64, max_value=128, step=16),
                     kernel_size=hp.Choice('conv_1_kernel_size', values = [3, 5]),
                     padding='same'))
    model.add(Conv2D(activation='relu',
                     filters=hp.Int('conv_2_filters', min_value=32, max_value=64, step=16),
                     kernel_size=hp.Choice('conv_2_kernel_size', values = [3, 5]),
                     padding='same'))
    model.add(Conv2D( activation='relu',
                      filters=hp.Int('conv_3_filters', min_value=32, max_value=128, step=16),
                      kernel_size=hp.Choice('conv_3_kernel_size', values = [3, 5]),
                      padding='same',
                      strides=2))
    model.add(Dropout(0.4))

    model.add(Conv2D(activation='relu',
                     filters=hp.Int('conv_4_filters', min_value=64, max_value=128, step=16),
                     kernel_size=hp.Choice('conv_4_kernel_size', values = [3, 5]),
                     padding='same'))
    model.add(Conv2D(activation='relu',
                     filters=hp.Int('conv_5_filters', min_value=64, max_value=128, step=16),
                     kernel_size=hp.Choice('conv_5_kernel_size', values = [3, 5]),
                     padding='same'))
    model.add(Conv2D(activation='relu',
                     filters=hp.Int('conv_6_filters', min_value=64, max_value=128, step=16),
                     kernel_size=hp.Choice('conv_6_kernel_size', values = [3, 5]),
                     padding='same',
                     strides=2))
    model.add(Dropout(0.4))

    model.add(Flatten())
    model.add(Dense(units=128, activation='relu'))
    model.add(Dropout(0.4))

    model.add(Dense(units=num_classes, activation='softmax'))

    model.compile(optimizer=keras.optimizers.Adam(hp.Choice('learning_rate', [1e-2, 1e-3, 1e-4])),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model
```

Рисунок 4.11. – Приклад реалізації Bayesian Optimization

```
Trial 5 Complete [00h 02m 59s]
val_accuracy: 0.015379472014804682

Best val_accuracy So Far: 0.9306252201398214
Total elapsed time: 00h 15m 33s
Results summary
Results in output/Hypertuning
Showing 10 best trials
Objective(name="val_accuracy", direction="max")

Trial 0 summary
Hyperparameters:
conv_1_filters: 128
conv_1_kernel_size: 3
conv_2_filters: 32
conv_2_kernel_size: 5
conv_3_filters: 112
conv_3_kernel_size: 3
conv_4_filters: 128
conv_4_kernel_size: 5
conv_5_filters: 128
conv_5_kernel_size: 3
conv_6_filters: 112
conv_6_kernel_size: 5
learning_rate: 0.0001
Score: 0.9306252201398214

Trial 1 summary
Hyperparameters:
conv_1_filters: 112
conv_1_kernel_size: 5
conv_2_filters: 64
conv_2_kernel_size: 5
conv_3_filters: 48
conv_3_kernel_size: 5
conv_4_filters: 112
conv_4_kernel_size: 5
conv_5_filters: 128
conv_5_kernel_size: 5
conv_6_filters: 96
conv_6_kernel_size: 3
learning_rate: 0.0001
Score: 0.9224339524904887

Trial 4 summary
Hyperparameters:
conv_1_filters: 128
conv_1_kernel_size: 5
conv_2_filters: 64
conv_2_kernel_size: 3
conv_3_filters: 96
conv_3_kernel_size: 5
conv_4_filters: 80
conv_4_kernel_size: 3
conv_5_filters: 112
conv_5_kernel_size: 3
conv_6_filters: 112
```

Рисунок 4.12. – Виведення оптимальних гіпперпарамтрів

Підставивши налаштовані гіперпараметри в нашу архітектуру (рис. 4.12), отримуємо досить гарні результати (рис. 4.13):

```
def _build_model(self):
    model = Sequential()

    model.add(Conv2D(input_shape=self.input_shape, activation='relu', filters=128, kernel_size=3, padding='same'))
    model.add(Conv2D(activation='relu', filters=32,kernel_size=5, padding='same'))
    model.add(Conv2D( activation='relu', filters=112,kernel_size=3,  padding='same'))
    model.add(Dropout(0.4))

    model.add(Conv2D(activation='relu', filters=128,kernel_size=5,padding='same',strides=2))
    model.add(Conv2D(activation='relu', filters=128, kernel_size=3,padding='same',strides=3))

    model.add(Conv2D(activation='relu',filters=112,kernel_size=5, padding='same'))

    model.add(Dropout(0.4))

    model.add(Flatten())
    model.add(Dense(units=128, activation='relu'))
    model.add(Dropout(0.4))

    model.add(Dense(units=self.num_classes, activation='softmax'))


    return model

def compile(self):
    self.model.compile(
        loss="categorical_crossentropy",
        # optimizer="adam",
        optimizer=keras.optimizers.Adam(learning_rate=0.0001),

        metrics=["accuracy"]
    )
```

Рисунок 4.13. – Архітектура підсумкової моделі

Модель починається із згорткового шару Conv2D, що приймає на вхід зображення з заданими параметрами форми (self.input_shape). На цьому шарі використовується 128 фільтрів з розміром ядра 3x3 та активаційною функцією 'relu'. Параметр padding='same' означає, що будуть додані нульові поля для того, щоб розмір вихідного зображення співпадав із розміром вхідного.

Далі ідуть два додаткові згорткові шари, що також використовують функцію активації 'relu', але вже з іншими параметрами фільтрів та розмірів ядра. Після них іде шар Dropout з параметром 0.4, який допомагає уникнути перенавчання, випадково відключаючи деякі нейрони під час тренування.

У наступному блоку знову йдуть три згорткові шари, вже з іншими параметрами та з використанням кроку (strides), який виконує функцію підвибірки, зменшуючи розмір вихідного зображення. Після цих шарів знову використовується Dropout.

Нарешті, після згорткових шарів йде шар Flatten, який перетворює багатовимірний тензор в одновимірний, після чого додається повнозв'язний шар (Dense) з 128 нейронами та активаційною функцією 'relu'. За ним знову йде шар Dropout, а потім - вихідний шар з кількістю нейронів, рівною числу класів (self.num_classes), і активаційною функцією 'softmax', що перетворює вихідні значення в ймовірності належності до кожного з класів.

Ми використовуємо Dropout з метою запобігання перенавчанню. Цей метод регуляризації відключає випадковий вибір нейронів під час процесу тренування. Такий підхід допомагає моделі не покладатися занадто сильно на конкретні нейрони і заохочує розподіл відповідальності серед усіх нейронів мережі.

Всі згорткові та повнозв'язні шари в моделі використовують активаційну функцію relu. Вона обнуляє всі від'ємні вхідні значення, що допомагає зменшити проблему зникаючого градієнта та прискорює процес навчання.

Після створення архітектури моделі ми компілюємо її, встановлюючи втрати, оптимізатор та метрики. Втрати вказують, яку функцію втрати модель повинна оптимізувати під час тренування. В нашому випадку ми використовуємо "categorical_crossentropy", яка є стандартною для багатокласової класифікації.

Ми використовуємо оптимізатор Adam з навчальною швидкістю 0.0001. Оптимізатор вказує, який алгоритм оптимізації слід використовувати для оновлення ваг моделі. Adam - це популярний вибір, оскільки він ефективно збалансовує швидкість навчання і стабільність.

Метрика "accuracy" використовується для відстеження точності класифікації під час тренування і перевірки.

Коли модель скомпільована, ми можемо навчати її за допомогою методу fit. Цей метод приймає тренувальні та тестові дані, кількість епох та розмір пакета.

Перевіримо модель на тестовому зображенні (рис. 4.3) з урахуванням усієї системи та отримаємо результати (рис. 4.14):

```
мета видалення шуму  мінімізувати бао повністю
усунути перешкоди для поліпшення якості вилучення
тексту шум на бзораженні може бнаувати різних
форм від випадкових пікселів різної кольору до
складніших фартеак іві гаких як нерівнос бао плями
на фотої фраіі шум може бу и спричинении різними
фак горамії включно з умовами зиомкиі якістю
камери бао процесом сканування документів
видалення може бути досягнуто за допомогою різних
гехнік алі орит міві включно з фморолої ічними
операціями інаприкладі ерозієюі розширеннямі
відкриттям їзакриттямі згладжуванням інаприкладі
розмит ям ауса бао медіанним фільт їром га іншими
спеціалізованими методамиі такими як усунення
шуму на основі статистичних методіві які ми
розглядали раніше
```

Рисунок 4.14. – Результати системи OCR

### 4.4. Telegram бот

Телеграм бот у цій системі використовується як зручний інтерфейс, що дає змогу користувачам взаємодіяти з системою розпізнавання тексту, мінімізуючи технічні знання, які зазвичай потрібні для роботи з такими системами.

Бот був розроблений з метою максимально спростити процес оптичного розпізнавання тексту для користувачів.

Основний сценарій використання бота включає кілька кроків:

1) Користувач натискає кнопку "Почати розпізнавання".
2) Бот відповідає: "Вітаємо в OCR Бот! Я допоможу вам розпізнати текст на ваших зображеннях. Будь ласка, спочатку оберіть модель, за допомогою якої будемо передбачати зображення: модель 1 чи модель 2."
3) Користувач обирає модель.
4) Бот підтверджує вибір моделі та просить завантажити зображення: "Ваша модель успішно обрана. Завантажте, будь ласка, зображення друкованого тексту в хорошій якості, на якому ви хочете розпізнати текст."
5) Користувач завантажує зображення.
6) Бот підтверджує отримання зображення: "Ваше зображення успішно завантажено. Зачекайте трохи, ваше зображення обробляється. Це може зайняти деякий час."
7) Після обробки бот повертає розпізнаний текст: "Ось ваш розпізнаний текст: ..."

Діаграма послідовностей на основі сценарію вище (рис.4.15):

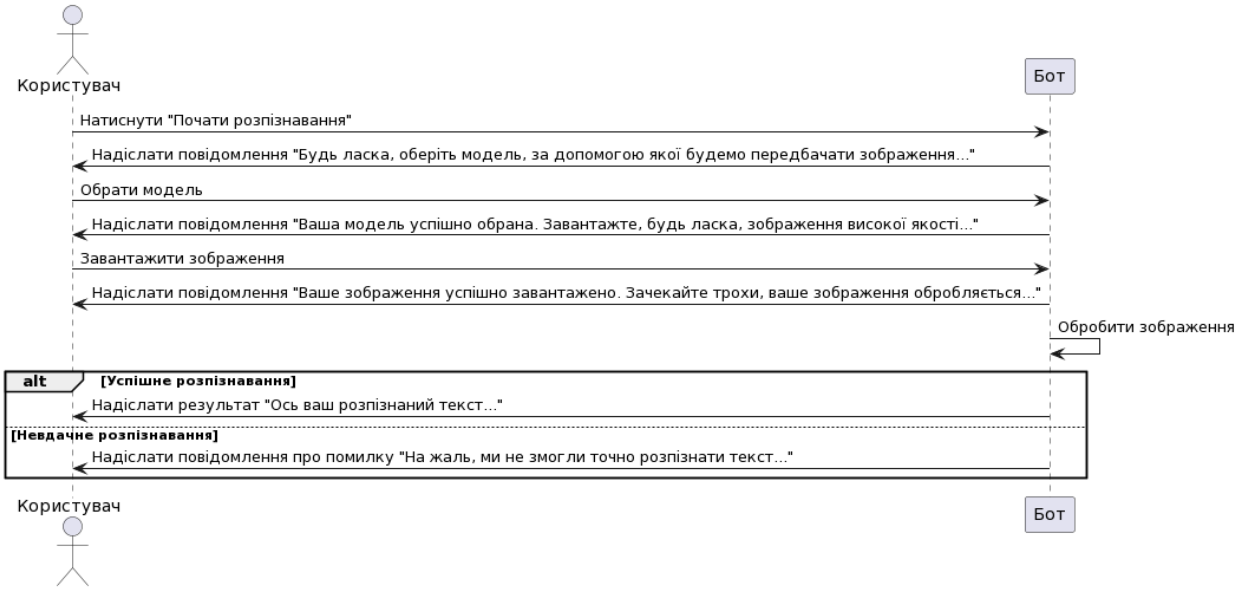



Рисунок 4.15. – Діаграма послідовностей

Use Case Diagram на основі сценарію вище зображено на рис. 4.16.

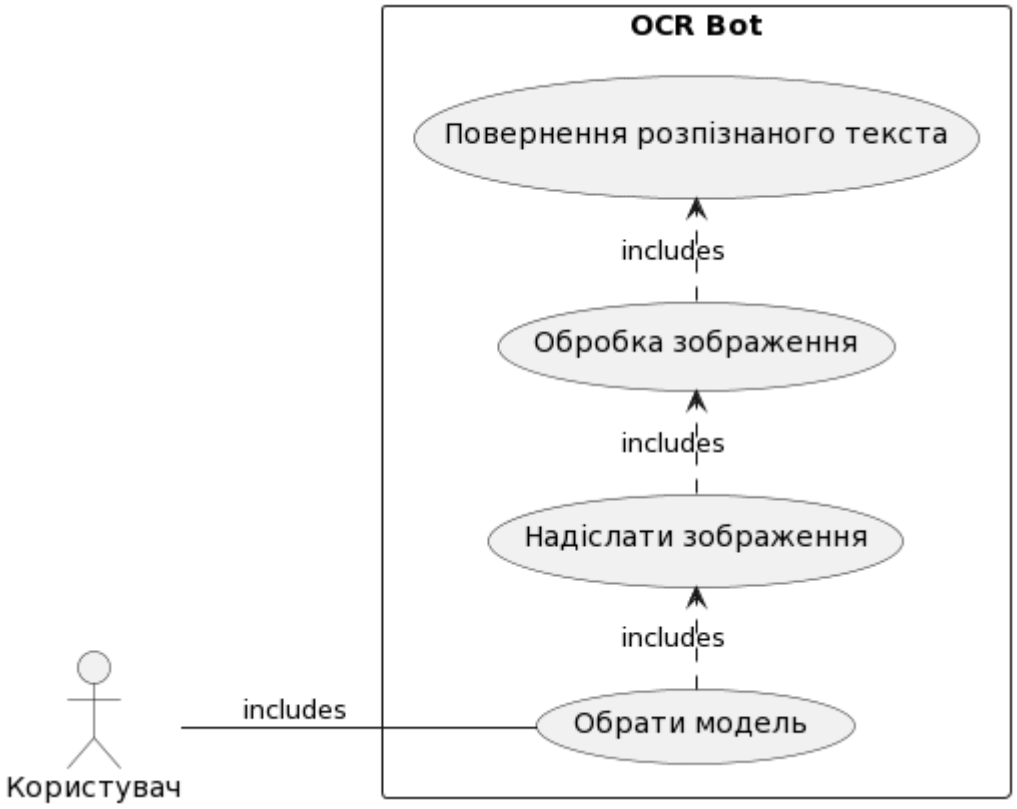



Рисунок 4.16. – Use Case Diagram

Стуктура бота має такий вигляд:

```
/handlers
 | /errors
 | /users
   | __init__.py
   | commands.py
   | ocr.py
   |start.py
 | __init__.py
 | texts.py
/keyboards
  |main.py
/middlewares
  |check_image.py
/states
  |ocr_steps.py
loader.py
main.py
```

Основою структури бота є різні модулі, кожен з яких відповідає за певну функцію. У директорії "/handlers" знаходяться оброблювачі команд, включно з оброблювачами помилок, командами та модулями оптичного розпізнавання тексту. Оброблювачі команд, зокрема, керують взаємодією бота з користувачем, відстежуючи та відповідаючи на різні команди, які вводить користувач.

У директорії "/keyboards" міститься основна клавіатура бота, яка забезпечує управління користувацьким інтерфейсом.

У директорії "/middlewares" знаходиться обробник перевірки зображення, який забезпечує якість завантажуваних зображень.

У директорії "/states" знаходяться стани бота, які визначають, як він має реагувати на різні ситуації та введення користувача.

Результати роботи бота бачимо на рис. 4.17, рис. 4.18, рис. 4.19.

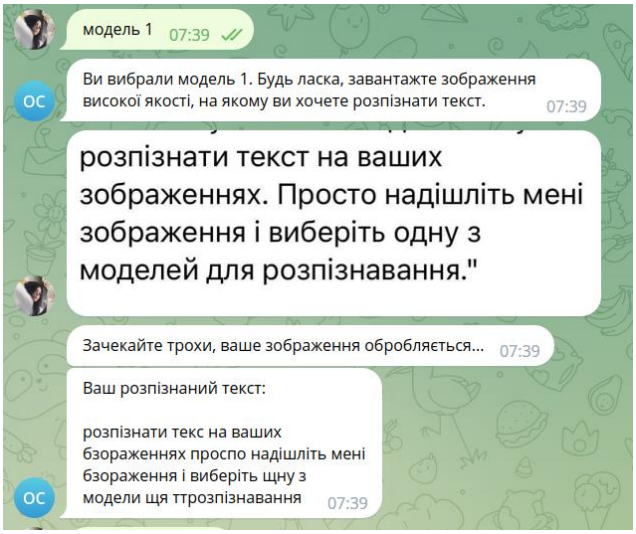



Рисунок 4.17. – Приклад роботи 1 моделі в боті

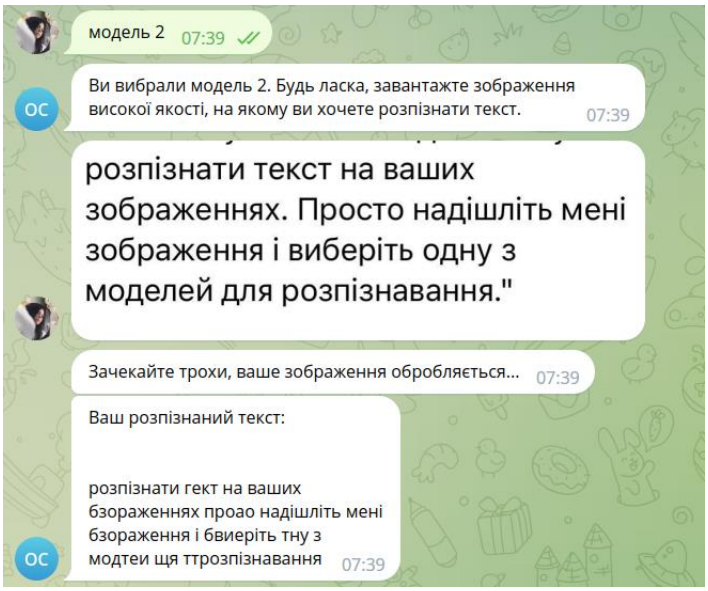



Рисунок 4.18. – Приклад роботи 2 моделі в боті

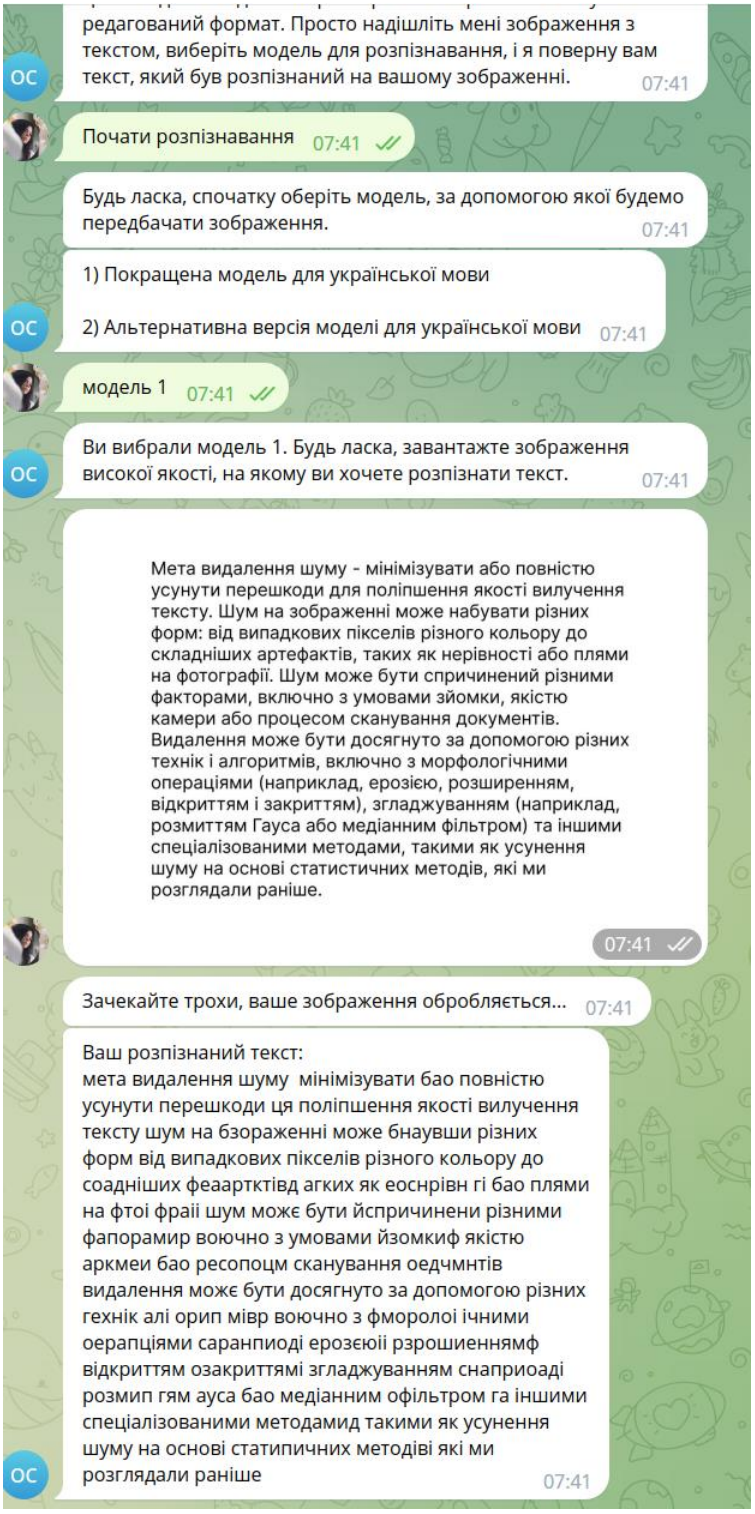



Рисунок 4.19. – Повний кейс роботи з ботом

# ВИСНОВОК ДО РОЗДІЛУ 4

У результаті цього розділу ми попередньо обробили дані, навчили модель сегментації тексту і дістали символи за допомогою ручних ознак. Для зручнішої візуалізації даних був розроблений бот для Телеграма.

Незважаючи на всі поліпшення моделі, система все ще далека від ідеалу. Вона допускає помилки досить часто. Виправити це можна надалі шляхом поліпшення моделі, збільшення набору даних або ж поліпшення сегментації символів, однак спочатку за основу брали примітивні моделі, тому надрезультатів на них все одно не вийде.

Можна за основу спробувати взяти моделі RNN або їхні поліпшені версії, щоб запам'ятовувався і контекст або ж взагалі використовувати трансформери, які є потужним інструментом.

Причина, через яку ми не використовували ці технології в даній роботі, в тому, що ці підходи вимагають величезних ресурсів в плані навчання і кількості даних, що навчаються.

Але навіть поточну модель можна поліпшити за допомогою постоброблення з використанням іншої моделі глибокого навчання, або ж використовувати словник і на основі нього виділяти відстань Левенштейна.

Так само важливо зазначити, що у вихідному датасеті містяться тільки символи українського алфавіту, тому для поліпшення якості, можна розширити датасет, враховуючи додаткові символи.

# ВИСНОВОК

У даній дипломній роботі було розглянуто різні підходи та методи оптичного розпізнавання тексту, а також розроблено власну систему OCR, якою можна скористатися через телеграм-бота або ж безпосередньо з коду.

В першому розділі були розглянуті можливі підходи і готові рішення даної задачі. У другому була представлена робота з датасетом. У третьому були описані деталі розробки системи з математичним поясненням всіх використаних підходів та алгоритмів. У четвертому були надані експерименти та запровадження налаштованої моделі в готовий продукт у вигляді Telegram бота.

Система все ще далека від ідеалу, але все базову функцію виконує правильно і стабільно. Остаточну модель, як говорилося у висновку до 4 розділу, можна поліпшити шляхом вибору кращої архітектури моделі, вибору більш розширеного датасету або ж використання пост опрацювання на основі відстані Левенштейна.

Ознайомитись детальніше з програмною реалізацією проєкту можна за наступним посиланням: https://github.com/lizonn/ocr_module.git

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Optical Character Recognition: What is It and How Does it Work [Guide] [Електронний ресурс] // Hmrishav Bandyopadhyay. – Режим доступу: https://www.v7labs.com/blog/ocr-guide (дата звернення: 01.03.2023).

2. Cloud Vision documentation [Електронний ресурс] // Google Cloud. – Режим доступу: https://cloud.google.com/vision/docs (дата звернення: 15.03.2023).

3. ABBYY FineReader [Електронний ресурс] // ABBYY. – Режим доступу: https://www.abbyy.com/ocr-sdk/features/adrt/ (дата звернення: 15.03.2023).

4. A Database of Printed Ukrainian Letters for Character Recognition [Електронний ресурс] // olekscode. – Режим доступу: https://github.com/olekscode/UkrainianOCR/tree/master (дата звернення: 20.05.2023).

5. Normalization (image processing) [Електронний ресурс] // Wikipedia. – Режим доступу: https://en.wikipedia.org/wiki/Normalization_(image_processing) (дата звернення: 16.04.2023).

6. Goyal, G., & Luthra, R. (2016). Skeleton generation for digital images based on performance evaluation parameters. International Journal of Signal Processing, Image Processing and Pattern Recognition, 9(2), 47-58.

7. Trier, Ø. D., Jain, A. K., & Taxt, T. (1996). Feature extraction methods for character recognition-a survey. Pattern recognition, 29(4), 641-662.

8. What's OCR? [Електронний ресурс] // Data ID Systems. – Режим доступу: http://www.dataid.com/aboutocr.htm (дата звернення: 30.05.2023).

9. Jain, A. K., Duin, R., & Mao, J. (2000). Statistical Pattern Recognition: A Review. IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(1), 4-37.

10. Mendel, G. (1993). Optical character recognition using morphological attributes. Louisiana State University and Agricultural & Mechanical College.

11. "Математична морфологія" [Електронний ресурс] // Горьков Олексій. – Режим доступу: https://habr.com/ru/articles/113626/ (дата звернення: 05.05.2023).

12. Арлазаров, В. В., Кляцкін, В. М., & Славін, О. А. (2015). Структурний аналіз текстових полів у системах потокового введення оцифрованих документів. Взято з http://www.isa.ru/proceedings/images/documents/2015-65-1/t-15-1_75-81.pdf

13. What Is Optical Character Recognition (OCR)? [Електронний ресурс] // Super.AI. – Режим доступу: https://super.ai/blog/what-is-optical-character-recognition-ocr (дата звернення: 10.05.2023).

14. Sattayakawee, N. (2013). Test scoring for non-optical grid answer sheet based on projection profile method. International Journal of Information and Education Technology, 3(2), 273.

15. Rodrigues, R. J., Thomé, A. C. G., & Carlos, A. (2000, August). Cursive character recognition–a character segmentation method using projection profile-based technique. In The 4th World Multiconference on Systemics, Cybernetics and Informatics SCI 2000 and The 6th International Conference on Information Systems, Analysis and Synthesis ISAS.

16. Convolutional Neural Networks cheatsheet [Електронний ресурс] // Stanford University. – Режим доступу: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks (дата звернення: 20.05.2023).

17. Basic Introduction to Convolutional Neural Network in Deep Learning [Електронний ресурс] // Pranshu Sharma. – Режим доступу: https://www.analyticsvidhya.com/blog/2022/03/basic-introduction-to-convolutional-neural-network-in-deep-learning/ (дата звернення: 20.05.2023).

18. Beginners Guide to Convolutional Neural Networks [Електронний ресурс] // Sabina Pokhrel. – Режим доступу: https://towardsdatascience.com/beginners-guide-to-understanding-convolutional-neural-networks-ae9ed58bb17d (дата звернення: 30.05.2023).

19. CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more… [Електронний ресурс] // Siddharth Das. – Режим доступу: https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5 (дата звернення: 26.05.2023).

20. A Conceptual Explanation of Bayesian Hyperparameter Optimization for Machine Learning [Електронний ресурс] // Will Koehrsen. – Режим доступу: https://towardsdatascience.com/a-conceptual-explanation-of-bayesian-model-based-hyperparameter-optimization-for-machine-learning-b8172278050f (дата звернення: 28.05.2023).

# ДОДАТОК 1

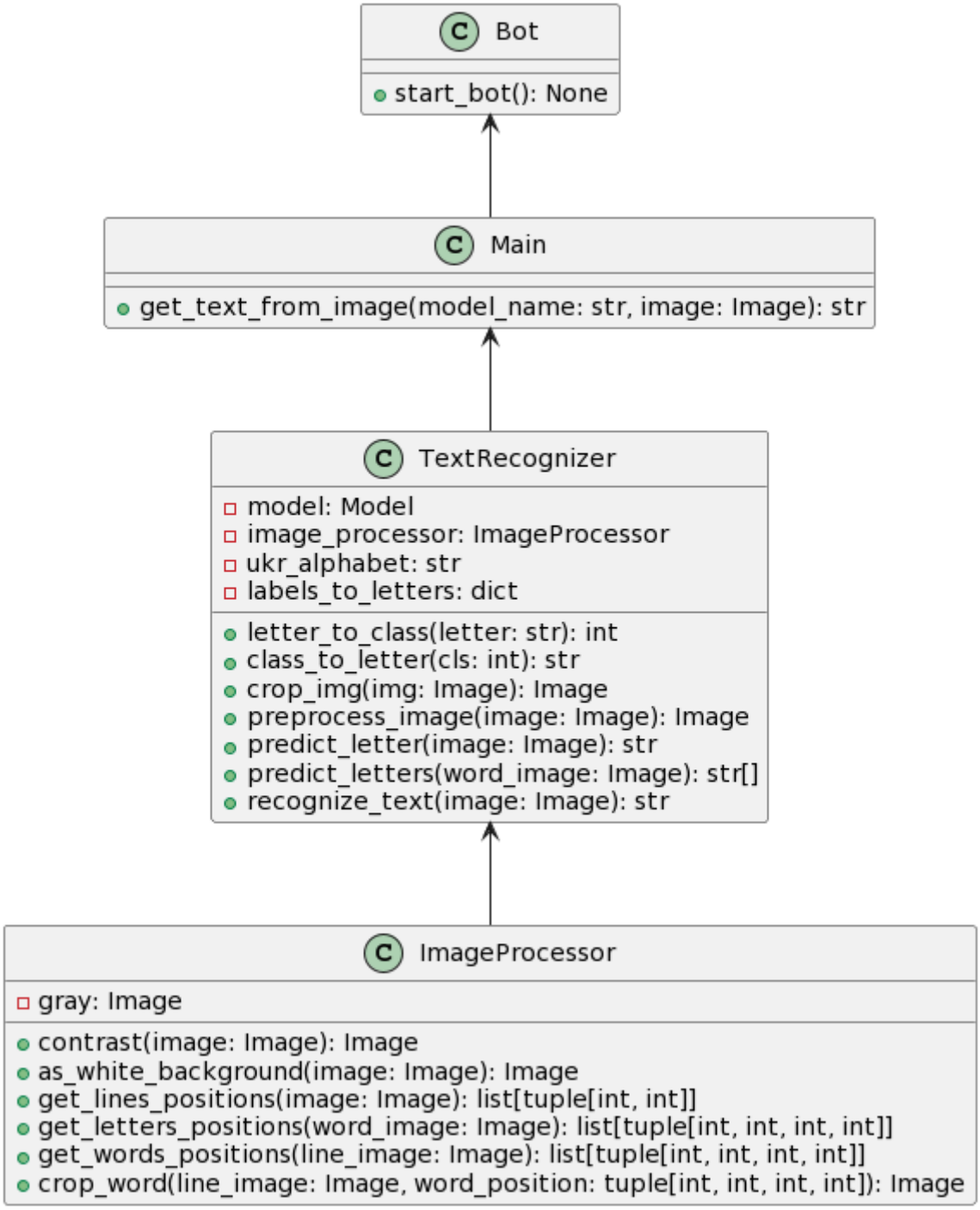
Система оптичного розпізнавання тексту

Діаграма класів (функціональна схема)

ІАЛЦ.467200.004 Д2

Аркушів 1

**Київ 2023 р**




| | | | | | |
|---|---|---|---|---|---|
| | | | | | ІАЛЦ.467200.004 Д1 |
| Зм. | Арк. | № докум. | Підпис | Дата | |
| Розробив | | Савенко Є.В. | | | **Система оптичного розпізнавання тексту** |
| Перевірив | | Кочура Ю.П. | | | Діаграма класів (функціональна схема) |
| Реценз. | | | | | |
| Н. Контр. | | Виноградов Ю.М. | | | |
| Затвердив | | | | | |

| Літ. | Аркуш | Аркушів |
|---|---|---|
| | 1 | 1 |

КПІ ім. Ігоря Сікорського, ФІОТ, ІО-з91

# ДОДАТОК 2

Система оптичного розпізнавання тексту

Діаграма активності (структурна схема)

ІАЛЦ.467200.005 Д2

Аркушів 1

**Київ 2023 р**


| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | ІАЛЦ.467200.005 Д2 | |
| Зм. | Арк. | № докум. | Підпис | Дата | | |
| Розробив | | Савенко Є.В. | | | **Система оптичного розпізнавання тексту** | Літ. / Аркуш 1 / Аркушів 1 |
| Перевірив | | Кочура Ю.П. | | | | |
| Реценз. | | | | | | КПІ ім. Ігоря Сікорського, ФІОТ, ІО-з91 |
| Н. Контр. | | Виноградов Ю.М. | | | Діаграма активності (структурна схема) | |
| Затвердив | | | | | | |

# ДОДАТОК З

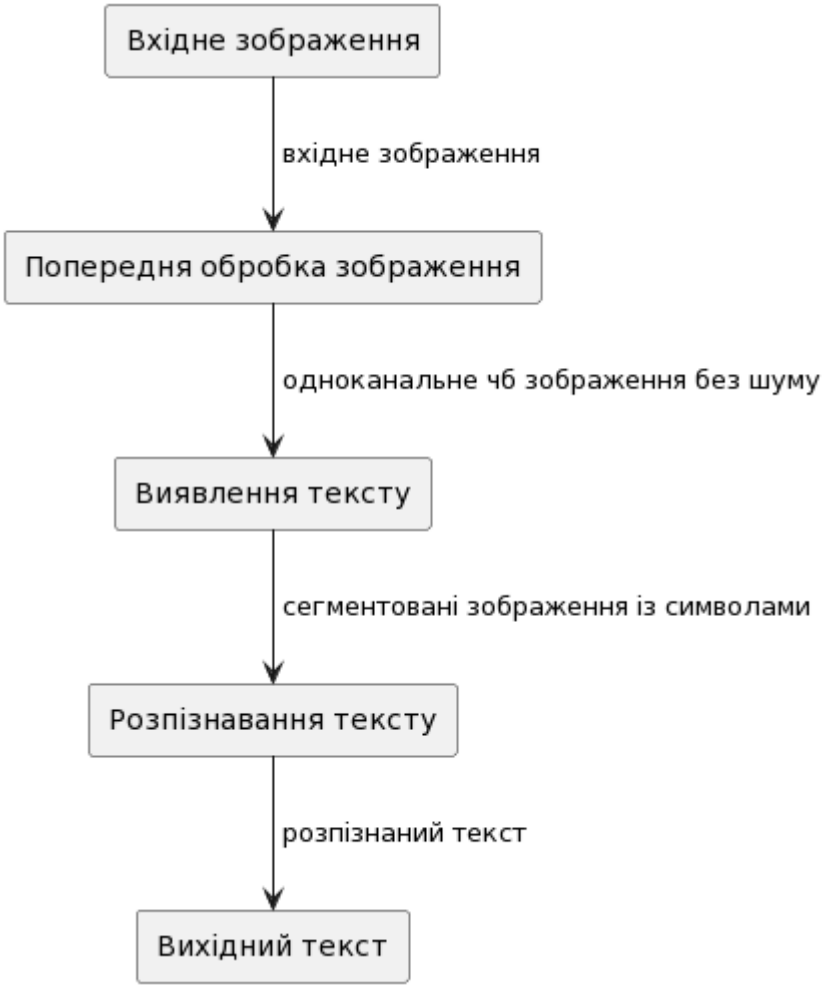
Система оптичного розпізнавання тексту

Діаграма потоку даних (принципова схема)

ІАЛЦ.467200.006 ДЗ

Аркушів 1

**Київ 2023 р**




| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | ІАЛЦ.467200.006 ДЗ | | |
| Зм. | Арк. | № докум. | Підпис | Дата | | | |
| Розробив | | Савенко Є.В. | | | **Система оптичного розпізнавання тексту** | Літ. | Аркуш 1 / Аркушів 1 |
| Перевірив | | Кочура Ю.П. | | | | | |
| Реценз. | | | | | Діаграма потоку даних (принципова схема) | КПІ ім. Ігоря Сікорського, ФІОТ, ІО-з91 | |
| Н. Контр. | | Виноградов Ю.М. | | | | | |
| Затвердив | | | | | | | |

# ДОДАТОК 4

Система оптичного розпізнавання тексту

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 20

**Київ 2023 р**

preprocessing_dataset.py

```
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from skimage import measure


def preprocess_image_db(img):
    coords = np.column_stack(np.where(img > 0))

    x_min, y_min = coords.min(axis=0)
    x_max, y_max = coords.max(axis=0)
    cropped_img = img[x_min:x_max+1, y_min:y_max+1]
    cropped_img = np.array(cropped_img, dtype=np.uint8)

    padded_img = cv2.copyMakeBorder(cropped_img, 1, 1, 1, 1, cv2.BORDER_CONSTANT, value=0)
    resized_img = cv2.resize(padded_img, (28, 28))
    return resized_img.flatten()



def preprocess_dataset(df):
    df_processed = df.copy()
    for i in range(len(df)):
        df_processed.iloc[i, 1:] = preprocess_image_db(df.iloc[i, 1:].values.reshape(28, 28)).flatten()

    return df_processed

if __name__ == '__main__':

    df = pd.read_csv("uaset.csv")

    processed_images = []
```

```
    for index, row in df.iterrows():
        try:
            image = row[1:].values.reshape([28, 28])
            processed_image = preprocess_image_db(image)
            processed_images.append(processed_image.flatten())
        except:
            print(index)

    processed_df = pd.DataFrame(processed_images, columns=df.columns[1:])
    processed_df.insert(0, 'label', df['label'])

    processed_df.to_csv("processed_uaset.csv", index=False)
```

## model.py

```
from keras import regularizers
from tensorflow.keras.layers import BatchNormalization
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.utils import to_categorical
from keras.optimizers import RMSprop
import keras

class CharacterRecognizer:
    def __init__(self, input_shape, num_classes):
        self.input_shape = input_shape
        self.num_classes = num_classes
        self.model = self._build_model()

    def _build_model(self):
        model = Sequential()

        model.add(Conv2D(input_shape=self.input_shape, activation='relu', filters=128, kernel_size=3,
padding='same'))
        model.add(Conv2D(activation='relu', filters=32, kernel_size=5, padding='same'))
        model.add(Conv2D(activation='relu', filters=112, kernel_size=3, padding='same'))
```

```
            model.add(Dropout(0.4))

            model.add(Conv2D(activation='relu', filters=128, kernel_size=5, padding='same', strides=2))
            model.add(Conv2D(activation='relu', filters=128, kernel_size=3, padding='same', strides=3))

            model.add(Conv2D(activation='relu', filters=112, kernel_size=5, padding='same'))
            model.add(Dropout(0.4))
            model.add(Flatten())
            model.add(Dense(units=128, activation='relu'))
            model.add(Dropout(0.4))

            model.add(Dense(units=self.num_classes, activation='softmax'))

            return model

        def compile(self):
            self.model.compile(
                loss="categorical_crossentropy",
                # optimizer="adam",
                optimizer=keras.optimizers.Adam(learning_rate=0.0001),

                metrics=["accuracy"]
            )

        def fit(self, X_train, y_train, X_test, y_test, epochs, batch_size=128):
            early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3)
            # self.model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs, validation_data=(X_test,
y_test), callbacks=[early_stopping])

    # для аугментованих даних
            self.model.fit(datagen.flow(X_train, y_train, batch_size=batch_size),
                           validation_data=(X_test, y_test),
                           callbacks=[early_stopping],
                           epochs=epochs)

        def save(self, path):
            self.model.save(path)
```

## train_model.py

```
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from keras.preprocessing.image import ImageDataGenerator

from ocr_module.models.model import CharacterRecognizer



df = pd.read_csv("dataset/processed_uaset.csv")

X = df[df.columns[1:]]
y = df['label']

# Нормалізація даних
X = X / 255.0

X = 1 - X

#  Розділяємо дані на навчальний і тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Змінюємо форму даних, щоб вони підходили для CNN
X_train = X_train.values.reshape(-1, 28, 28, 1)
X_test = X_test.values.reshape(-1, 28, 28, 1)

# Перетворюємо мітки в one-hot вектори
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)



class ImageDataGeneratorWithNoise(ImageDataGenerator):
    def __init__(self, noise_stddev=0.1, **kwargs):
        super().__init__(**kwargs)
        self.noise_stddev = noise_stddev
```

```
        def random_transform(self, x):
            x = super().random_transform(x)
            noise = np.random.normal(scale=self.noise_stddev, size=x.shape)
            return np.clip(x + noise * 0.2, 0., 1.)


    # новий генератор з шумом
    datagen = ImageDataGeneratorWithNoise(
        zoom_range=0.1,
        height_shift_range=0.1,
        noise_stddev=0.1
    )

    datagen.fit(X_train)

    recognizer = CharacterRecognizer(input_shape=(28, 28, 1), num_classes=y_train.shape[1])
    recognizer.compile()
    recognizer.fit(X_train, y_train, X_test, y_test, epochs=15)
recognizer.save('best_model.h5')
```

## preprocessing.py

```
    import cv2
    import numpy as np
    import matplotlib.pyplot as plt
    from skimage import measure

    class ImageProcessor:
        def __init__(self):
            pass

        def load_image(self, path):
            self.img = cv2.imread(path)
            return self.img

        def to_gray(self, img):
            return  cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
def contrast(self, img):
    img = img.copy()

    min_val = np.min(img)
    max_val = np.max(img)

    pixel_range = max_val - min_val

    img = (img - min_val) * (255.0 / pixel_range)
    img = np.round(img).astype(np.uint8)

    return img


def as_white_background(self, im):
    colors = np.array(im).flatten()
    median = np.percentile(colors, 50)
    if median > 128:
        return im
    else:
        return 255 - im


def get_histogram(self,im, horiz=False):
    if horiz:
        return np.array(im).mean(axis=0)
    else:
        return np.array(im).mean(axis=1)

def get_hor_histogram(self,im):

    return np.array(im).sum(axis=(0, 1))


def get_edges(self, im):
    if len(im.shape) > 2:
```

```
            im = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)

        edges = cv2.Canny(im, 100, 200)

        return edges

    def get_lines_positions(self, im, threshold=.002):
        hist = self.get_histogram(self.get_edges(im))
        threshold = hist.min() + threshold * (hist.max() - hist.min())

        height = hist.shape[0]
        lines = []

        pos = 0
        while True:
            while pos < height and hist[pos] < threshold: pos += 1
            if pos == height:
                break

            line_start = pos
            while pos < height and hist[pos] >= threshold: pos += 1
            line_end = pos
            lines.append((line_start, line_end))
            if pos == height:
                break

        lines = [i for i in lines if abs(i[0] - i[1]) > 5]
        return lines

    def get_words_positions(self,line, threshold=0.1, space_threshold=0.5):
        if len(line.shape) > 2:
            line = cv2.cvtColor(line, cv2.COLOR_BGR2GRAY)

        hist = self.get_histogram(line, horiz=True)
        threshold = hist.max() - threshold * (hist.max() - hist.min())

        h_hist = self.get_histogram(line)
```

```
        letters_height = h_hist / h_hist.mean() / 2
        letters_height = 1 - np.round(letters_height)
        letters_height = letters_height.sum()
        space_threshold = int(round(letters_height * space_threshold))
        width = hist.shape[0]
        words = []

        pos = 0
        while True:
            while pos < width and hist[pos] >= threshold: pos += 1
            if pos == width:
                break

            word_start = pos
            while pos < width and hist[pos] < threshold: pos += 1
            word_end = pos
            new_word = (word_start, word_end)
            if len(words) != 0:
                prev_words = words[-1]
                if word_start - prev_words[1] < space_threshold:
                    words[-1] = (words[-1][0], word_end)
                else:
                    words.append(new_word)
            else:
                words.append(new_word)

            if pos == width:
                break

        words = [i for i in words if abs(i[0] - i[1]) > 5]
        return words

    def crop_word(self, img, word):
        start,end = word
        cropped_word = img[:, start:end]

        return cropped_word
```

```
    def get_letters_positions(self, word_image):
        gray = cv2.cvtColor(word_image, cv2.COLOR_BGR2GRAY)
        _, threshold = cv2.threshold(gray, 150, 255, cv2.THRESH_BINARY_INV)
        kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3,3))
        dilated = cv2.dilate(threshold, kernel, iterations=1)

        labels = measure.label(dilated, connectivity=2, background=0)
        positions = []

        for label in np.unique(labels):
            if label == 0:
                continue

            mask = np.zeros(dilated.shape, dtype="uint8")
            mask[labels == label] = 255

            contours, _ = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

            for contour in contours:
                x, y, w, h = cv2.boundingRect(contour)
                if w > 2 and h > 10:
                    positions.append((x, y, w, h))

        return positions


if __name__ == '__main__':

    processor = ImageProcessor()
    image = cv2.imread('images/noise.jpg')

    processed_image = processor.contrast(image)
    processed_image = processor.as_white_background(processed_image)
    lines = processor.get_lines_positions(processed_image)
```

```
            # words = processor.get_words_positions(processed_image)



            plt.imshow(processed_image,  cmap='gray')
            plt.title('Підготовлене зображення')
            plt.show()

            # виводить позиції рядків
    print("Line positions:", lines)
```

## text_recognition.py

```
        from ocr_module.preprocessing import ImageProcessor
        import cv2
        import numpy as np
        from tensorflow.keras.models import load_model



        class TextRecognizer:
            def __init__(self, model, image_processor):
                self.model = model
                self.image_processor = image_processor
                self.ukr_alphabet =
'АБВГҐДЕЄЖЗИІЇЙКЛМНОПРСТУФХЦЧШЩЬЮЯабвгґдеєжзиіїйклмнопрстуфхцчшщьюя0987654321!"#
%’()*+,-./:;'
                self.labels_to_letters = {i: self.ukr_alphabet[i] for i in range(len(self.ukr_alphabet))}


            def letter_to_class(self, letter):
                return self.ukr_alphabet.index(letter)


            def class_to_letter(self, cls):
                return self.labels_to_letters[cls]


            def crop_img(self, img):
                if img is None or img.size == 0:
                    raise ValueError("Invalid input image")


                coords = np.column_stack(np.where(img > 0))
```

```
        if coords.size == 0:
            raise ValueError("No non-zero pixels found")

        # обчислюємо обмежувальний прямокутник для ненульових координат і обрізаємо зображення
        x_min, y_min = coords.min(axis=0)
        x_max, y_max = coords.max(axis=0)
        cropped_img = img[x_min:x_max + 1, y_min:y_max + 1]

        if cropped_img.size == 0:
            raise ValueError("Cropped image is empty")

        # Обчислюємо нові розміри зображення, зберігаючи співвідношення сторін
        height, width = cropped_img.shape[:2]
        if height > width:
            new_height = 28
            new_width = int(width * new_height / height)
        else:
            new_width = 28
            new_height = int(height * new_width / width)

        resized_img = cv2.resize(cropped_img, (new_width, new_height))

        if resized_img.size == 0:
            raise ValueError("Resized image is empty")

        # ширина і висота білих смуг, які потрібно додати
        padding_height = 28 - new_height
        padding_width = 28 - new_width

        if padding_height < 0 or padding_width < 0:
            raise ValueError("Padding is negative. Check the resized image dimensions.")

        top_border_height = padding_height // 2
        bottom_border_height = padding_height - top_border_height

        left_border_width = padding_width // 2
```

```
        right_border_width = padding_width - left_border_width

        padded_img = cv2.copyMakeBorder(resized_img, top_border_height, bottom_border_height,
                                        left_border_width, right_border_width, cv2.BORDER_CONSTANT, value=255)
        padded_img = cv2.bitwise_not(padded_img)

        return padded_img

    def preprocess_image(self, image):
        image = cv2.resize(image, (28, 28))

        if len(image.shape) > 2 and image.shape[2] > 1:
            image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        image = np.reshape(image, (28, 28, 1))

        image = image / 255.0
        return image

    def predict_letter(self, image):
        # plt.imshow(image)
        # plt.show()

        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        image = self.preprocess_image(image)
        prediction = self.model.predict(np.array([image]))
        predicted_class = np.argmax(prediction)
        predicted_letter = self.class_to_letter(predicted_class)

        return predicted_letter

    def predict_letters(self, word_image):
        letters_images_resized = []
        letters_positions = self.image_processor.get_letters_positions(word_image)

        for pos in letters_positions:
            x, y, w, h = pos
```

```
            letter_img = word_image[y:y + h, x:x + w]

            letter_img = cv2.cvtColor(letter_img, cv2.COLOR_BGR2GRAY)

            letter_img = self.crop_img(letter_img)
            letter_img = 255 - letter_img
            letter_img = self.preprocess_image(letter_img)

            if len(letter_img.shape) > 2 and letter_img.shape[2] == 3:
                gray_img = cv2.cvtColor(letter_img, cv2.COLOR_BGR2GRAY)
            else:
                gray_img = letter_img

            letters_images_resized.append(gray_img[..., np.newaxis])

        if letters_images_resized:
            letters_images_array = np.stack(letters_images_resized, axis=0)
        else:
            return ''

        predictions = self.model.predict(letters_images_array)

        predicted_classes = [np.argmax(p) for p in predictions]
        predicted_letters = [self.class_to_letter(c) for c in predicted_classes]

        return predicted_letters

    def recognize_text(self, image):

        self.image_processor.gray = image
        self.image_processor.gray = self.image_processor.contrast(self.image_processor.gray)
        self.image_processor.gray = self.image_processor.as_white_background(self.image_processor.gray)
        lines = self.image_processor.get_lines_positions(self.image_processor.gray)

        text = ""
        for line in lines:
            start, end = line
```

```
                line_image = self.image_processor.gray[start:end]

                words = self.image_processor.get_words_positions(line_image)

                for word in words:
                    word_image = self.image_processor.crop_word(line_image, word)

                    letter_predictions = self.predict_letters(word_image)
                    word_text = "".join(letter_predictions)
                    text += word_text + " "
                text += "\n"
            return text.lower()


        if __name__ == '__main__':
            model = load_model('models/ua_model.h5')
            processor = ImageProcessor()
            recognizer = TextRecognizer(model, processor)

            image = cv2.imread('images/noise.png')

            res = recognizer.recognize_text(image)

    print(res)
```

## main.py

```
        from ocr_module.preprocessing import ImageProcessor
        from ocr_module.text_recognition import TextRecognizer
        import cv2
        import numpy as np
        import matplotlib.pyplot as plt
        from tensorflow.keras.models import load_model

        def get_text_from_image(model_name,image):

            model = load_model(model_name)
            processor = ImageProcessor()
```

```
        recognizer = TextRecognizer(model, processor)


        res = recognizer.recognize_text(image)

        # print(res)
        return res

    if __name__ == '__main__':
        model_name = 'models/ua_model.h5'
        image_path = 'images/noise.png'
        image = cv2.imread(image_path)
        res = get_text_from_image(model_name, image)
print(res.lower())
```

## start.py

```
from aiogram import types
from aiogram.dispatcher.filters.builtin import CommandStart
from bot.loader import dp
from bot.handlers import texts

from bot.keyboards.main import main_kb

@dp.message_handler(CommandStart(),state='*')
async def start_with_deeplink(message: types.Message,state):

    await state.reset_state(with_data=False)

await message.answer(texts.GREETING_TEXT,reply_markup=main_kb)
```

## commands.py

```
from aiogram import types
from aiogram.dispatcher.storage import FSMContext

from bot.handlers import texts
from bot.loader import dp
from bot.states.ocr_steps import Form
from bot.keyboards.main import main_kb,model_choice_kb
```

```
@dp.message_handler(commands="help", state="*")
async def help(message: types.Message, state: FSMContext):
    await message.answer(texts.HELP,reply_markup=main_kb)
    await state.reset_state(with_data=False)


@dp.message_handler(commands='image', state="*")
@dp.message_handler(text=texts.START_BUTTON)
async def image(message: types.Message, state: FSMContext):
    await message.answer(texts.CHOISE_MODEL)
    await message.answer(texts.CHOISE_MODEL2, reply_markup=model_choice_kb)

  await Form.waiting_for_model_choice.set()
```

**ocr.py**

```
import aiohttp
from aiogram import types
from aiogram.dispatcher.storage import FSMContext
from matplotlib import pyplot as plt

from bot.handlers import texts
from bot.loader import dp,bot
from bot.states.ocr_steps import Form
from bot.keyboards.main import model_choice_kb
from bot.middlewares.check_image import image_quality_check

import io
import cv2
import numpy as np
from PIL import Image
from bot import config
from ocr_module.main import get_text_from_image

VALID_MODELS = [texts.MODEL1,texts.MODEL2]
```

```
@dp.message_handler(state=Form.waiting_for_model_choice)
async def process_model_choice(message: types.Message, state: FSMContext):
    chosen_model = message.text

    data = await state.get_data()
    attempts = data.get('attempts', 0)

    if chosen_model not in VALID_MODELS:
        attempts += 1
        await state.update_data(attempts=attempts)
        if attempts >= 2:
            chosen_model = VALID_MODELS[1]  # автоматически выбирается модель 2
            await message.answer('Ви не правильно ввели модель. Було автоматично вибрано друга модель.',reply_markup=types.ReplyKeyboardRemove())
        else:
            await message.answer('Ви ввели неправильну модель. Будь ласка, введіть назву моделі ще раз.',reply_markup=model_choice_kb)
            return
    else:
        await message.answer(
            'Ви вибрали ' + chosen_model + '. Будь ласка, завантажте зображення високої якості, на якому ви хочете розпізнати текст.',reply_markup=types.ReplyKeyboardRemove())

    await state.update_data(chosen_model=chosen_model)
    await Form.waiting_for_image.set()


@dp.message_handler(content_types=['text', 'photo', 'document'], state=Form.waiting_for_image)
@image_quality_check
async def process_image(message: types.Message, state: FSMContext, image_path=None, **kwargs):

    await message.answer(texts.WAIT_MODEL)
    data = await state.get_data()
    chosen_model = data.get('chosen_model')


    # TODO можна винести потім в окрему функцію
    file_path = await bot.get_file(image_path)
```

```
            file_url = f"https://api.telegram.org/file/bot{config.API_TOKEN}/{file_path.file_path}"

            async with aiohttp.ClientSession() as session:
                async with session.get(file_url) as resp:
                    file_body = await resp.read()

            image = Image.open(io.BytesIO(file_body))

            if chosen_model == texts.MODEL1:
                model_name = 'ocr_module/models/best_model.h5'
            else:
                model_name = 'ocr_module/models/ua_model.h5'

            res = get_text_from_image(model_name, image)

            await message.answer(f'Ваш розпізнаний текст:\n{res}')

    await state.finish()
```

## texts.py

```
        GREETING_TEXT = 'Вітаємо у OCR Бот! Я допоможу вам розпізнати текст на ваших зображеннях. Просто надішліть мені зображення і виберіть одну з моделей для розпізнавання.'

        HELP = 'OCR Бот - це інструмент, який допомагає розпізнати друкований текст на скріншотах. ' \
               'Ви можете використовувати цей бот для швидкого перетворення зображень тексту в редагований формат. ' \
               'Просто надішліть мені зображення з текстом, виберіть модель для розпізнавання, і я поверну вам текст, ' \
               'який був розпізнаний на вашому зображенні. '

        # START_OCR = 'Будь ласка, завантажте зображення друкованого тексту в хорошій якості, на якому ви хочете розпізнати текст'

        CHOISE_MODEL = 'Будь ласка, спочатку оберіть модель, за допомогою якої будемо передбачати зображення.'

        CHOISE_MODEL2 = '''1) Покращена модель для української мови

        2) Альтернативна версія моделі для української мови'''

        START_BUTTON = 'Почати розпізнавання'
```

```
MODEL1 = "модель 1"
MODEL2 = "модель 2"
WAIT_MODEL = 'Зачекайте трохи, ваше зображення обробляється...'
```

## Keyboards/main.py

```
from aiogram.types import ReplyKeyboardMarkup,  KeyboardButton
from bot.handlers import texts
b1 = KeyboardButton(texts.START_BUTTON
main_kb = ReplyKeyboardMarkup(resize_keyboard=True)
main_kb.add(b1)

model1 = KeyboardButton(texts.MODEL1)
model2 = KeyboardButton(texts.MODEL2)

model_choice_kb = ReplyKeyboardMarkup(resize_keyboard=True)
model_choice_kb.add(model1)
model_choice_kb.add(model2)
```

## states/ocr_steps.py

```
from aiogram.dispatcher.filters.state import StatesGroup, State

class Form(StatesGroup):
  waiting_for_model_choice = State()
  waiting_for_image = State()
```

## loader.py

```
from aiogram import Bot, Dispatcher, types
from aiogram.contrib.fsm_storage.memory import MemoryStorage

from bot import config

bot = Bot(token=config.API_TOKEN,
        parse_mode=types.ParseMode.HTML)

dp = Dispatcher(bot, storage=MemoryStorage())
```

## bot/main.py

```
from bot.handlers import dp
from bot.loader import bot

from aiogram.types import (BotCommand,
```

```
                    BotCommandScopeAllPrivateChats)

    from aiogram import executor

    def start_bot(dp):
        user_commands = [
            BotCommand('image', 'Розпізнати нове зображення'),
            BotCommand('help', 'Інструкція як користуватися ботом')
        ]

        async def startup(dp):
            await bot.set_my_commands(
                commands=user_commands,
                scope=BotCommandScopeAllPrivateChats())

        executor.start_polling(dp, on_startup=startup)


    if __name__ == '__main__':
start_bot(dp)
```