Національний університет «Полтавська політехніка імені Юрія Кондратюка»
(повне найменування вищого навчального закладу)
Навчально-науковий інститут інформаційних технологій та робототехніки
(повна назва факультету)
Кафедра комп'ютерних та інформаційних технологій і систем
(повна назва кафедри)

Пояснювальна записка
до дипломного проекту (роботи)
магістра
(освітньо-кваліфікаційний рівень)
на тему

Програмне забезпечення для розпізнавання тексту із зображень

Виконав: студент 6 курсу, групи 601-ТН
спеціальності
122 Комп'ютерні науки
(шифр і назва напряму)
Олепір Є. В.
(прізвище та ініціали)
Керівник Митрофанов П. Б.
(прізвище та ініціали)

Полтава – 2024 року

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИНАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ПОЛТАВСЬКА ПОЛІТЕХНІКА ІМЕНІ ЮРІЯ КОНДРАТЮКА»

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА РОБОТОТЕХНІКИ

КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА
спеціальність 122 «Комп'ютерні науки»

на тему

«Програмне забезпечення для розпізнавання тексту із зображень»

Студента групи 601-ТН Олепіра Євгенія Володимировича

Керівник роботи
кандидат технічних наук,
доцент Митрофанов П. Б.

Завідувач кафедри
кандидат технічних наук,
доцент Двірна О.А.

Полтава – 2024

РЕФЕРАТ

Кваліфікаційна робота магістра: 90 с., 39 рисунків, 1 додаток, 26 джерел.

Об’єкт дослідження: процеси розпізнавання тексту із зображень, включаючи попередню обробку зображень, виявлення тексту, розпізнавання символів та постобробку отриманих результатів, втілені в програмному забезпеченні.

Мета роботи: розробка програмного забезпечення для розпізнавання тексту із зображень, включаючи попередню обробку зображень, виявлення тексту, розпізнавання символів та постобробку отриманих результатів.

Методи: аналіз існуючих рішень, об'єктно-орієнтоване програмування, обробка зображень, розпізнавання символів.

Ключові слова: PYTHON, ЗОБРАЖЕННЯ, СИМВОЛИ, PYTESSERACT, OCR.

ANNOTATION

Qualification work of master's degree: 90 p., 39 figures, 1 application, 26 sources.

Object of study: processes of text recognition from images, including image preprocessing, text detection, character recognition and post-processing of the obtained results, implemented in software.

The goal of the work: to develop software for text recognition from images, including image preprocessing, text detection, character recognition, and post-processing of the results.

Methods: аналіз існуючих рішень, об'єктно-орієнтоване програмування, обробка зображень, розпізнавання символів.

Keywords: PYTHON, IMAGES, SYMBOLS, PYTESSERACT, OCR..

ЗМІСТ

СПИСОК УМОВНИХ СКОРОЧЕНЬ

ПЗ – програмне забезпечення.

РТЗ – розпізнавання тексту з зображень.

ОРС – оптичне розпізнавання символів (англ. OCR – Optical Character Recognition).

БД – база даних.

API – інтерфейс програмування додатків (англ. Application Programming Interface).

ІС – інформаційна система.

GUI – графічний інтерфейс користувача (англ. Graphical User Interface).

ML – машинне навчання (англ. Machine Learning).

CNN – згорткова нейронна мережа (англ. Convolutional Neural Network).

TF – тензорний фреймворк (англ. TensorFlow).

NN – нейронна мережа.

ПЗО – програмне забезпечення для обробки.

ПК – персональний комп’ютер.

ВСТУП

Через стрімкий розвиток інформаційних технологій та цифрових інструментів велика кількість текстової інформації залишається у форматі зображень або сканованих документів, що ускладнює її обробку та аналіз. Програмне забезпечення для розпізнавання тексту із зображень (OCR – Optical Character Recognition) є ключовою технологією для автоматизації цього процесу, дозволяючи перевести текст з паперових або зображувальних носіїв у цифровий формат для подальшого зберігання, редагування та аналізу.

Актуальність теми дослідження обумовлена зростаючою потребою у швидкій та точній обробці великого обсягу текстових даних у різних галузях – від архівування документів до обробки даних у медичній, юридичній та освітній сферах. Використання OCR-систем дозволяє значно підвищити продуктивність роботи з документами, зменшити витрати часу на ручне введення тексту, а також знизити ризик помилок. Однак, попри значний прогрес у розвитку технологій OCR, досі існують труднощі, пов'язані з розпізнаванням текстів на низькоякісних зображеннях, текстів у нестандартних шрифтах або різними мовами.

Проблематика дослідження полягає у необхідності розробки ефективного програмного забезпечення для розпізнавання тексту, яке б могло успішно працювати з текстами у складних умовах, таких як нерівномірне освітлення, спотворення, наявність шумів на зображенні або низька роздільна здатність. Також актуальними є питання підвищення точності розпізнавання багатомовних текстів, складних шрифтів та різних графічних елементів, що інтегровані у текстовий контекст. Постобробка отриманих результатів і корекція помилок залишаються критично важливими для досягнення якісного кінцевого результату.

Труднощі на сьогодні включають питання адаптації OCR-систем до різних типів документів та мов, а також удосконалення алгоритмів для зменшення похибок розпізнавання. Використання машинного навчання та

глибоких нейронних мереж у розробці OCR-програм дозволяє досягати високих результатів, але ці методи потребують великих обсягів даних для тренування моделей та значних обчислювальних ресурсів. Одним з викликів є забезпечення високої швидкості роботи програмного забезпечення при збереженні точності.

Метою дослідження є розробка програмного забезпечення для розпізнавання тексту із зображень, яке забезпечить високу точність розпізнавання, ефективну постобробку результатів та можливість роботи з текстами на різних мовах та форматах. Програма повинна бути здатна працювати з різноманітними зображеннями, включаючи зображення низької якості, та забезпечувати мінімізацію помилок у розпізнаванні символів та слів.

Об'єктом дослідження є процеси розпізнавання тексту із зображень, включаючи попередню обробку зображень, виявлення тексту, розпізнавання символів та постобробку отриманих результатів.

Суб'єктом дослідження є програмне забезпечення для розпізнавання тексту із зображень, яке реалізує сучасні алгоритми машинного навчання для підвищення точності розпізнавання та корекції помилок.

Таким чином, дослідження спрямоване на розробку ефективної OCR-системи, яка відповідатиме сучасним вимогам до автоматизації обробки текстової інформації з різних джерел, забезпечуючи високу продуктивність та точність роботи навіть у складних умовах.

РОЗДІЛ І
ТЕОРЕТИЧНІ ЗАСАДИ РОЗПІЗНАВАННЯ ТЕКСТУ ІЗ ЗОБРАЖЕНЬ

1.1. Огляд існуючої літератури про розпізнавання тексту із зображень

Існує багато досліджень в області обробки зображень і розпізнавання образів, які пов'язані з розпізнаванням рукописних символів. Деякі зосереджуються на розпізнаванні англійських рукописних символів в автономному режимі з використанням нейронної мережі. Видалення шуму здійснюється за допомогою медіанного фільтра, бінаризація – за допомогою глобального методу Оцу, виявлення країв – за допомогою фільтра Собела, розширення та заповнення також виконуються як частина попередньої обробки. На етапі виділення особливостей використовується метод діагонального виділення особливостей. Розбиваємо покращене зображення на 54 рівні зони. Таким чином, з кожного символу отримують 54 ознаки. Для класифікації використовується нейронна мережа з прямим зворотним поширенням. Діагональні ознаки забезпечують високу точність розпізнавання порівняно зі звичайними горизонтальними та вертикальними методами виділення ознак. Використання цієї системи на основі 54 ознак дало ефективність розпізнавання 98%.

S.V. Rajashekararadhya запропонували систему вилучення ознак на основі центроїда зображення та центроїда зони для розпізнавання рукописних цифр для чотирьох популярних південноіндійських писемностей. Цими мовами є малаялам, тамільська, каннада та телугу [1].

На етапі попередньої обробки було виконано шумозаглушення, корекцію нахилу, нормалізацію та проріджування. На етапі виділення ознак використовуються центроїд зображення, центроїд зони та гібридні методи. Для подальшої класифікації та розпізнавання використовуються нейронна мережа зі зворотним поширенням та класифікатори найближчого сусіда. Точність розпізнавання, отримана для чисел каннада і телугу, становить 99%.

Для числівників тамільською та малаяламською мовами отримано 96% та 95% відповідно.

S.L.Mhetre запропонували два різних підходи для розпізнавання рукописних чисел деванагарі. У першому методі використовуються сіткові ознаки. У другому методі виділяються ознаки ICZ (Image Centroid Zone) та ZCZ (Zone Centroid Zone) на основі інформації про відстань. Тут для класифікації використовуються ШНМ та оцінка відповідності, а також оцінюється точність, отримана за допомогою двох підходів. При класифікації ШНМ забезпечує кращу точність порівняно з оцінкою відповідності.

S. V. Rajashekararadhya описали систему виділення ознак на основі зон (метод ICZVDDICZHRD) для розпізнавання рукописних цифр/змішаних цифр південно-індійського письма. Для класифікації використовуються нейронна мережа найближчого сусіда, нейронна мережа прямого поширення та машинні класифікатори опорних векторів. За допомогою машини опорних векторів отримано 98,65 % розпізнавання для чисел каннада, 96,1 % для тамільських чисел, 98,6 % для чисел телугу та 96,5 % для малаяламських чисел [1].

S. V. Rajashekararadhya також описали метод виділення кутових ознак на основі центроїдної зони зображення (ICZ) для розпізнавання рукописних цифр шрифту каннада. Обчислюється центроїд зображення цифри, а потім зображення ділиться на n рівних зон. Обчислюється середній кут від центроїда символу до пікселів, присутніх у зоні. Ця процедура повторюється послідовно для всіх зон, присутніх на зображенні цифр. Врешті-решт виділяється n таких ознак. Для класифікації використовуються класифікатор найближчого сусіда та машини опорних векторів. Точність розпізнавання досягла 96,05% для цифр каннада з використанням машин опорних векторів.

Гіта Сінха подбав про розпізнавання арабських цифр. На етапі попередньої обробки використовуються бінаризація, дилатація, ерозія, видалення шуму та нормалізація. Було використано три методи виділення ознак: центроїдна зона зображення (ICZ), центроїдна зона (ZCZ) та гібридний метод виділення ознак. Гібридні методи виділення ознак є комбінацією

ICZ+ZCZ. Для класифікації використовується SVM класифікатор. Швидкість розпізнавання становить 97,21% для рукописної арабської цифри.

Seema A. Dongare запропонував розпізнавання символів деванагарі поетапно: попередня обробка документа, сегментація з використанням лінійної сегментації, сегментація слів і символів, виділення ознак з використанням зонного підходу з подальшим розпізнаванням за допомогою нейронної мережі прямого поширення. Розпізнавання рукописних ієрогліфів деванагарі є досить складною задачею через наявність широрекхи, сполучних символів та схожості форм для декількох ієрогліфів. Тут зроблено спробу підвищити точність і продуктивність розпізнавання

Гіта Сінха представив розпізнавання рукописних символів Gurumukhi. Етап попередньої обробки включає такі кроки, як перетворення шкали сірого, бінаризація за методом Оцу, фільтрація та морфологічна операція, видалення шуму, скелетування, виявлення перекосу. Для вилучення ознак використовується зональна методика, а для розпізнавання рукописних символів гурумухі – SVM-класифікатор. Отримана точність розпізнавання становить 95,11% [1].

Сандіп Саха запропонував 40-точкову екстракцію ознак для розпізнавання англійських рукописних символів за допомогою багатошарової нейронної мережі прямого поширення. Все зображення розбивається на 16 зон, а потім обчислюється середня інтенсивність кожної зони. Потім все зображення ділиться по діагоналі зліва зверху вниз, справа зверху вниз, зліва знизу вгору і справа знизу вгору і виділяються внутрішні ознаки клітинок.

Нарешті, вектори ознак, що складаються з 40 ознак, тестуються за допомогою штучної нейронної мережі, і повідомляється, що вони мають кращу ефективність розпізнавання.

Сангіта Сасідхаран описує сегментацію розпізнавання рукописних символів малаяламською мовою в режимі офлайн. Етап попередньої обробки включає видалення шуму та бінаризацію. На етапі сегментації використовується лінійна сегментація з використанням методу профілю

горизонтальної проекції. Сегментація символів фокусується на сегментації недоторканих символів, сегментації приголосних, що дотикаються до Valli (спеціальний малаяламський символ) та сегментації приголосних, що дотикаються до Chandrakala (спеціальний малаяламський символ). Ефективність, отримана в цій роботі, становить 94,08%.

Аніта Пал запропонував трасування границь разом з дескриптором Фур'є для розпізнавання рукописних англійських символів. На етапі попередньої обробки виконується скелетування та нормалізація. На етапі виділення ознак, виявлення границь виконується за допомогою методу 8-ми сусідніх сусідів. Для класифікації використовується нейронна мережа.

В роботі Reetika Verma описано виділення ознак прибою та нейронну мережу, яка продемонструвала здатність вирішувати складні задачі розпізнавання символів. Етап попередньої обробки включає видалення шуму та покращення зображення. Для виділення та класифікації ознак використовується метод прибою та нейронна мережа. Цей метод є швидким, недорогим і дозволяє отримати більш точний результат.

У роботі Abdul Rahiman M запропоновано систему розпізнавання рукописних символів, засновану на алгоритмі позиційного аналізатора вертикальних і горизонтальних ліній. На етапі попередньої обробки для видалення шуму використовується медіанний фільтр. На етапі сегментації використовується розділення рядків і символів, що дає ізольований символ.

Ознаки виділяються на основі кількості та положення горизонтальних і вертикальних ліній. Для класифікації використовується класифікатор на основі дерева рішень. Точність розпізнавання становить 91%.

У роботі Pranchi Mukherji запропоновано методику вилучення ознак форми для розпізнавання рукописних символів. Попередня обробка включає видалення шуму за допомогою фільтра Гауса, бінаризацію за методом Осту, скелетонізацію. Для методу виділення ознак використовується алгоритм усередненого стисненого кодування напрямку штрихів. Для класифікації

використовується класифікатор Decision Tree. Загальна точність розпізнавання становить 86.4%.

Порівняння різних літературних джерел, згаданих вище, підсумовано в Таблиці 1.1 [1].

Таблиця 1.1 – Порівняння літературних джерел про розпізнавання символів

Автор	Попередня обробка	Сегментація	Виділення ознак	Класифікація	Точність розпізнавання
1	2	3	4	5	6
J. Pradeep	Видалення шуму, бінаризація, виділення країв, дилатація і заповнення	-	Діагональні ознаки	Нейронна мережа з зворотним поширенням помилок	98%
S. V. Rajashekararadhya	Видалення шуму, корекція нахилу, нормалізація, скелетизація	-	Зональний центр зображення, зональний центр гібридного зображення	Нейронна мережа з зворотним поширенням помилок, найближчий сусід	99% для каннада, 96% для тамільської, 95% для малаялам
S. L. Mhetre	Конвертація кольору в сірий, видалення шуму, порогова обробка, скелетизація, нормалізація розміру	-	Метод на основі сітки, методи ICZ та ZCZ	Штучна нейронна мережа та метод порівняння оцінок	-
S. V. Rajashekararadhya	Видалення шуму, корекція нахилу, нормалізація, скелетизація	-	Зональний центр зображення, зональний центр гібридного зображення	Метод опорних векторів (SVM)	98.65% для каннада, 98.6% для телугу, 96.1% для тамільської, 96.5% для малаялам
S. V. Rajashekararadhya	Видалення шуму, корекція нахилу, нормалізація, скелетизація	-	Метод зонального кута	Метод опорних векторів (SVM)	96.05%

Продовження Табл. 1.1

1	2	3	4	5	6
Gita Sinha	Бінаризація, дилатація, ерозія, видалення шуму, нормалізація	-	Зональний центр зображення, зональний центр гібридного зображення	Метод опорних векторів (SVM)	97.21%
Seema A. Dongare	Конвертація кольору в сірий, видалення шуму, бінаризація	Лінія, слово, символ	Зональний центр зображення, зональний центр гібридного зображення	Штучна нейронна мережа (ANN)	-
Gita Sinha	Конвертація кольору в сірий, видалення шуму, бінаризація, фільтрація, згладжування контурів, виявлення нахилу, скелетизація	Лінія, слово, символ	Зональний центр зображення, зональний центр гібридного зображення	Метод опорних векторів (SVM)	95.11%
Sandeep Saha	Конвертація кольору в сірий, бінаризація	Обрізання зображення	40 точкових ознак	Штучна нейронна мережа (ANN)	-
Sangeetha Sasidharan	Видалення шуму, бінаризація	Сегментація ліній за допомогою проекційного профілю, сегментація символів неперервних символів	-	-	94.08%
Anita Pal	Скелетизація, нормалізація	-	Фур'є-ознаки (метод 8 сусідів)	Багатошарова нейронна мережа (MLP)	94%
Reetika Verma	Видалення шуму	-	Виділення ознак Surf	Нейронна мережа з зворотним поширенням помилок	-

Крім того, організація поштових марок Сполучених Штатів використовує оптичне розпізнавання символів для розпізнавання поштових

відправлень з початку 60-х років, що ґрунтується на основі технологічних розробок, головним чином, плідного винахідника Джейкоба Рейнбоу. У 1965 році вони почали планувати перевести всю банківську систему США на цю технологію. цю технологію, крок, який у майбутньому зробив революцію в системі оплати рахунків у Великій Британії. Великобританії.

Потім у 70-х роках в Індії були зроблені кроки доктором Сінгха в Індійському технологічному інституті в Харагпурі. А систематична система аналізу вирівнювання для розпізнавання мови гінді представлена в докторській дисертації Сінгхи.

Інша розробка такого типу систем була ініційована Палітом і Чаудхурі, або ми знаємо їх як Паліт і Чаудрі. Команда, що складається з професора Б. Б. Чаудхарі, У. Паліта, М. Мітре та У. Гараєна з Індійського технологічного інституту в Делі Індійського технологічного інституту в Делі, винайшла єдиний у своєму роді ринковий ресурс або продукт для друкованої системи розпізнавання гінді в Індії. Подібний Аналогічний тип інструменту було надано Центру розробок для Advanced Computation (CDAC) у 2000 році для комерційного продажу продукту і продавався під назвою «Chitrankan». Методологія базується на корекції Широрекхи, наведеній Чаудхарі та Полом, за винятком того. що кут нахилу таких заголовків говорить про кут нахилу повного документа. Спочатку виконуються взаємопов'язані компоненти в документі з роботою. Вимога похилої точки приймається шляхом додавання значної кількості ребер між рівнем і лініями, що з'єднують головні пікселі лівого фрагмента рядка та останні пікселі правильних фрагментів рядка відрізків кожної з ліній. Існує не так багато архівів, відомих як мульти нахиленими записами, в яких задані рядки вмісту не схожі один на одного. Ця методика є розширенням наведеної для оцінки нахилу в багатонахилених звітах. нахилів у багатонахилених звітах [2].

Діас і Чандра додатково запропонували для вмісту автономну процедуру оцінки нахилу, яка була більш надійною на основі математичної морфології. Дас і Чаудхарі запропонували ще один технічний термін для ідентифікації та

сегментації символів гінді, які були надруковані машиною. надрукованих машиною. Дас і Чаудхарі запропонували іншу техніку для сегментації ідентифікаційних та надрукованих машинним способом символів хінді.

Загалом, для розпізнавання символів і тексту з зображень, обчислюється центр символів, а потім пронумерована фотографія або фотографія символів розбивається на чверть рівних зон. Вводиться розташування в середній точці від внутрішньої частини персонажа до пікселів, присутніх у сегменті зони.

Ці кроки в порядку або в рядку було повторення для всіх боксів, присутніх для зональної площі. У мережі майже не залишиться вертикальних блоків, що мають вільні передні пікселі. Отже, оцінка висвітлення цих решіток або зональної ділянки у векторах висвітлення дорівнює нулю.

Цю методику потроху перероблено для всієї зони, присутньої на цифровому або символьному зображенні. Для отримання перспективи можна використовувати методику «половина на половину», засновану на зонах або матриці, в різних друкованих стилях.

Розпізнавання транскрибованого змісту є відокремленим типом арабських мовних символів повністю здійснюється як гіпотетично, так і по суті тестових робіт з певним коефіцієнтом розпізнавання, що було очевидно. У звіті про дослідження наведено результати розпізнавання, включаючи презентацію підробленої нейронної системи, підготовку та тестування точності та ефективності розпізнавання, які визначаються рамками безладу, час підготовки ШНМ та кількість віків для різних випадків вилучення елементів в режимі онлайн та вилучення від'єднаних компонентів, а також проведення дослідження подібності між усіма розглянутими випадками.

Етапи попередньої обробки, які планують вносити свій внесок у процедуру розпізнавання символів в Інтернеті та поза мережею, додатково покращують вилучення елементів в Інтернеті та поза мережею на тій підставі, що вони підвищують якість та виділяють основні елементи зображення – якщо має місце позамережева попередня обробка та постпідготовка, а також перебіг подій в Інтернеті у випадку попередньої обробки в Інтернеті. У цьому завданні

попередня обробка включає два етапи або два види, які відомі як попередня обробка в Інтернеті та відключена попередня обробка [3].

Таким чином, машина OCR намагається відсканувати зображення, щоб розділити відскановані фотографії на частини, а набрані дані визначають, які слова або цифри з'явилися на темних і світлих ділянках зображення. На початку роботи системи розпізнавання порівнюють ці зображення зі збереженими растровими зображеннями певних шрифтів, наприклад, Times New Roman тощо.

Результати спроб і випробувань таких систем розпізнавання символів або розпізнавання образів створили їм репутацію неточних і неефективних систем. Системи розпізнавання символів нового покоління використовують різні алгоритми нейромережевих технологій для визначення або спостереження краю нахилу, лінії безперервності та розриву за межами написаних символів, а також зворотного боку тексту. Беручи до уваги нерівності ручки та паперу, чорнила, а потім система розпізнавання робить припущення про символ.

Система OCR усереднює всі значення з різних машинних кодів, щоб отримати єдине рішення для зчитування. Система розпізнавання може розуміти велику кількість шрифтів різних мов, однак розпізнавання рукописного тексту різної форми та розміру все ще залишається проблемою для системи, тому виникає потреба в додатковій нейронній потужності штучної мережі. Інженери думають про різні способи покращення розпізнавання мови та рукописного тексту.

Як обговорювалося вище, ми можемо використовувати нейронні мережі для усунення цього обмеження. Нейронні системи корисні в тому випадку, якщо у нас є величезний набір даних для підготовки та навчання нейронної системи. Набори даних – одна з найважливіших речей при створенні іншої нейронної системи. Без легітимного набору даних підготовка нейронної системи є марною.

Існує також приказка про підготовку фальшивої нейронної системи: «Waste in, refuse out» (з англ. «Відходи всередину, сміття назовні»). Це означає, що інформація, яку надано для набору даних, щоб підготувати результат, буде саме такою. За таких умов, як ми отримаємо результат? Після того, як зображення буде відфільтровано, проведемо обчислення обробки характеристик, які вилучать з зображення значущі характеристики і перенесемо їх у набір даних або, краще, в державну базу даних. Імпортовані ознаки матимуть числові характеристики і, як правило, будуть розміщені в кластерах. Маючи ці якості, можна підготувати нейронну систему і отримати пристойний кінцевий продукт. Проблема навколо охарактеризованих наборів даних додатково полягає у свідомо обраних розрахункових характеристиках. Тонкощі мають велике значення і можуть суттєво вплинути на результати та кінцевий результат [4].

Найбільш важливими з них для методологій нейронних систем є наступні, як продемонстровано додатково за допомогою схеми на Рис. 1.1:

1. Негативна картина зображення, де вхідні дані дорівнюють нулю для чорного і одиниці для білого, значення між цими 2 значеннями говорять про релевантність пікселів.
2. Позиція координати X, відраховує пікселі по лівій стороні зображення, а також по середині від найменшої прямокутної сітки, яка може бути зроблена з розпізнаванням всіх «1» пікселів в сітці.
3. Позиція координати Y, відраховує пікселі від найнижчої точки зображення у сітці
4. Розмір між пікселями сітки.
5. Пік між пікселями сітки.
6. Повна площа «1» пікселя в межах розпізнаного зображення.
7. Сумарне положення «1» пікселя по координаті X відносно середини сітки. Ця точка може мати значення -ve, якщо зображення є лівим, як у випадку з літерою L.

8. Середнє значення координати Y цілих «1» пікселів, пов'язаних з центром маси сітки і поділене на пік сітки.

9. Середньоквадратичне значення довжини пікселя координати X, виміряне з точністю до півдюжини. Ця інформація може мати високу ціну для фотографій, у яких 9 пікселів квадратного розміру розділені в межах координати Y, як це буде для W і M.

10. Середньоквадратичне значення довжини пікселів координати Y обчислюється з точністю до семи значень.

11. Середній результат рівномірного та вертикального розділення для кожного «1» пікселя, як оцінено у верхніх авансах. Це включає в себе хорошу ціну для ліній від кута до кута, які рухаються зліва направо, і – ве вартість для похилих ліній, які рухаються з одного боку в інший бік.

12. Середнє значення для квадратної координати X відстань налаштовує різницю координати Y на кожний «1» піксель. Це дає залежність дисперсії координати Y від координати X.

13. Середнє квадратичне відстаней координати Y від координати X для кожного «1» пікселя.

14. Сума цих точок з координатою Y для кутів, що зустрічаються, як розраховано в попередньому пункті. Цей тип функції може забезпечити високу ціну, якщо є додаткові кути у верхній частині сітки, як в межах «Y» [5].

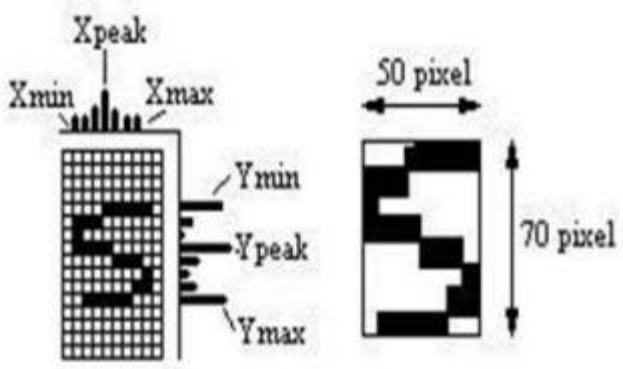

Рисунок 1.1 – Приклад визначення символів із зображення

1.2. Основні поняття розпізнавання тексту із зображень (OCR)

Розпізнавання тексту (OCR) набуло широкого застосування на початку 1990-х років для оцифрування історичних газет. Відтоді воно зазнало численних удосконалень. Сучасні рішення використовують найсучасніші технології для оптимізації робочих процесів обробки документів.

До появи розпізнавання текстів цифрове форматування документів передбачало трудомісткий передрук тексту. Це забирало багато часу і призводило до неточностей і помилок.

OCR перетворює відскановані документи, PDF-файли або зображення на дані, які можна редагувати та шукати. Він аналізує форми, візерунки та розташування символів у документі й перетворює їх на машинозчитуваний текст.

Оптичне розпізнавання символів (OCR) – це трансформаційна технологія, яка перетворює різні типи документів, такі як відскановані паперові документи, PDF-файли або зображення, зроблені цифровою камерою, в машинозчитувані і придатні для редагування дані. Використовуючи OCR, текст з друкованих, надрукованих або рукописних документів можна розпізнати і оцифрувати, зробивши його доступним для пошуку і використання в різних додатках.

Системи розпізнавання тексту аналізують форму і структуру символів на зображенні документа і зіставляють їх з відповідним текстом у заздалегідь визначеному наборі символів. Цей процес включає складні алгоритми та методи розпізнавання образів для точного перетворення візуального представлення тексту в цифровий формат, який можна редагувати, шукати та обробляти за допомогою комп'ютера [6].

На практиці OCR дозволяє компаніям і приватним особам швидко та ефективно перетворювати паперову інформацію в цифрову, полегшуючи оцифрування та управління документами. Це особливо корисно для таких завдань, як архівування, введення даних і пошук інформації, де перетворення

фізичних документів у цифрову форму може значно підвищити ефективність і доступність. Розпізнавання тексту широко використовується в різних галузях, зокрема у фінансовій, медичній, юридичній та державній сферах, де управління та обробка великих обсягів документів є критично важливими.

Основні етапи, необхідні для розпізнавання тексту із зобрадень:

Крок 1: Попередня обробка зображення (див. Рис. 1.2). Коли документ відскановано або зроблено знімок, програма розпізнавання тексту виконує попередню обробку зображення, щоб підвищити його якість і читабельність. Це може включати такі завдання, як зменшення шуму, вирівнювання зображення та підвищення контрастності для покращення чіткості тексту.

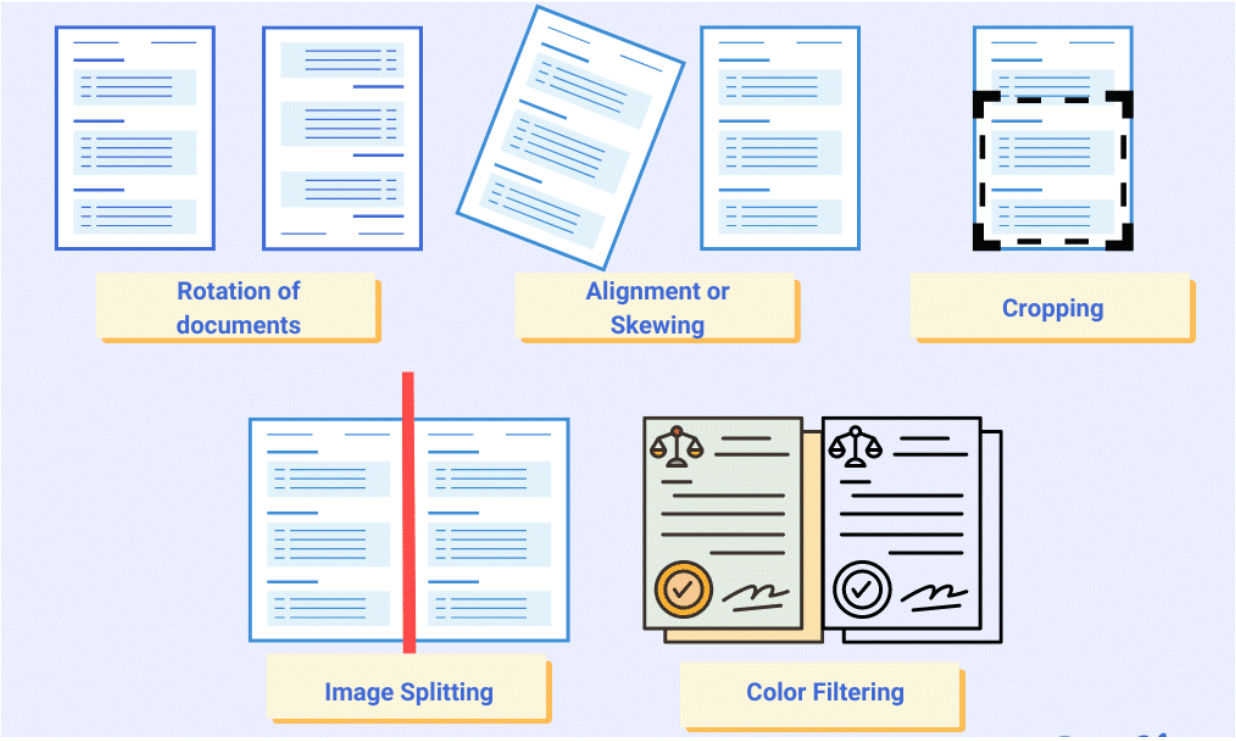

Рисунок 1.2 – Попередня обробка зображення

Попередня обробка зображення для OCR є важливим етапом, який суттєво впливає на точність та якість розпізнавання тексту. Основна мета цієї стадії полягає у підготовці зображення таким чином, щоб алгоритми OCR могли ефективно й правильно ідентифікувати символи та структуру тексту. Вона включає кілька важливих кроків.

Перший крок – це масштабування зображення або зміна його розміру, щоб збалансувати якість тексту й оптимізувати обробку системою OCR. Зображення може бути зменшене або збільшене в залежності від його початкової роздільної здатності.

Другим важливим етапом є бінаризація, під час якої зображення перетворюється з кольорового або сірого в бінарне (чорно-біле). Це дозволяє алгоритмам OCR чітко відрізнити текст від фону. Для цього використовують методи порогової обробки, такі як метод Оцу або адаптивне порогування. Вибір методу залежить від особливостей зображення – рівномірності освітлення, контрасту, наявності тіней або спотворень.

Далі проводиться видалення шумів і артефактів. Цей крок особливо важливий для зображень низької якості, сканованих документів або фотографій тексту. Видалення шуму досягається за допомогою фільтрів, таких як медіанний або гауссовий, які допомагають усунути зайві деталі, зберігаючи при цьому чіткість символів.

Наступним етапом є вирівнювання зображення. Оскільки текст на зображенні може бути нахиленим або перекошеним, це може створювати проблеми для OCR. Для виправлення цього використовується процес обертання, що полягає в автоматичному визначенні й корекції нахилу тексту на зображенні.

Контраст та яскравість зображення також можуть бути скориговані для покращення видимості тексту. Це робиться для того, щоб символи стали більш чіткими й легко розпізнаваними системою. Зазвичай ці зміни налаштовуються автоматично на основі аналізу зображення.

Ще один крок у попередній обробці – це сегментація. Вона полягає у розбитті зображення на окремі блоки тексту, рядки та навіть окремі символи, якщо це необхідно. Мета сегментації – полегшити роботу OCR, забезпечуючи точніше розпізнавання.

Після завершення цих етапів зображення стає оптимізованим для роботи з OCR, що підвищує точність розпізнавання тексту [7].

Крок 2: Виявлення тексту (див. Рис. 1.3). Алгоритми розпізнавання тексту аналізують попередньо оброблене зображення для виявлення областей, що містять текст. Цей процес включає виявлення шаблонів і фігур, які нагадують символи, слова та абзаци в документі.

Рисунок 1.3 – Виявлення тексту

Виявлення тексту на зображенні є ключовим етапом підготовки до оптичного розпізнавання символів (OCR), який забезпечує автоматичне знаходження текстових блоків на зображенні перед їх обробкою. Основна мета цього процесу – точне визначення областей, які містять текст, щоб підвищити ефективність і точність подальших етапів розпізнавання.

Процес виявлення тексту починається з аналізу зображення для виділення потенційних текстових областей. Один із перших методів – це використання крайових детекторів, таких як оператор Собеля або Кенні. Вони допомагають виявити контури об'єктів на зображенні, у тому числі контури символів та слів. В результаті цих операцій створюється контурна карта зображення, яка вказує на можливі місця розташування тексту.

Після цього зазвичай застосовують методи порогової обробки або бінаризації для покращення видимості тексту і відділення його від фону. Це

дозволяє підкреслити текстові області, які можна виділити для подальшого аналізу.

Для більш точного виявлення тексту широко використовують методи, засновані на кластеризації, такі як алгоритм згорткових нейронних мереж (CNN). Ці алгоритми дозволяють навчати модель розпізнавати текстові області на зображеннях з високим рівнем точності. Такі моделі можуть ідентифікувати не тільки стандартні текстові блоки, але й складніші форми тексту, як-от вигнутий або різнокольоровий текст на зображенні.

Сегментація є наступним важливим кроком у процесі виявлення тексту. Вона полягає в розбитті виявлених текстових областей на окремі блоки, рядки або навіть символи, якщо це потрібно. Це особливо важливо для зображень, де текст розміщений нерівномірно, або присутні різні шрифти та розміри. Сучасні системи використовують такі методи, як алгоритми прямокутної прив'язки або методи регіональних пропозицій (RPN), які дозволяють ефективно виділяти окремі блоки тексту.

Важливим елементом є і корекція геометрії. Якщо текст на зображенні нахилений або спотворений, це може негативно вплинути на точність розпізнавання. Для вирішення цієї проблеми застосовують алгоритми вирівнювання, що автоматично коригують кут нахилу тексту (deskewing), а також алгоритми перспективної трансформації, якщо текст зображений під кутом.

Останній етап виявлення тексту – це фільтрація результатів, яка дозволяє видалити помилкові або невірні виявлення. Зазвичай це здійснюється шляхом порівняння виявлених областей з моделями тексту, що використовуються в алгоритмах машинного навчання.

Загалом, процес виявлення тексту забезпечує ефективну і точну ідентифікацію текстових областей на зображенні, що є важливим етапом перед розпізнаванням тексту системою OCR [7].

Крок 3: Розпізнавання символів (див. Рис. 1.4). Після того, як текстові області ідентифіковано, програма розпізнавання виконує розпізнавання символів, аналізуючи форми та шаблони окремих символів. Це передбачає порівняння візуальних особливостей кожного символу із заздалегідь визначеним набором шаблонів або статистичних моделей, щоб визначити найімовірніший збіг символів.

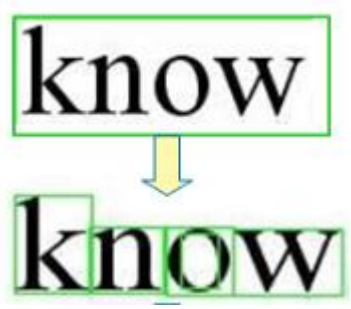

Рисунок 1.4 – Розпізнавання символів

Розпізнавання символів (OCR) – це ключовий етап у перетворенні зображень із текстовою інформацією в машинно-читабельний формат. На цьому етапі система аналізує кожен виділений текстовий блок або символ, визначений під час попередніх етапів, і намагається перетворити його в конкретну літеру, цифру або інший символ.

Процес розпізнавання символів починається з сегментації зображення на окремі символи або групи символів. Це дозволяє системі працювати з окремими елементами тексту і підготувати їх до розпізнавання. Сегментація зазвичай виконується після виявлення текстових областей і забезпечує правильне розбиття тексту на блоки, рядки та символи.

Далі система використовує різні методи для розпізнавання символів. Традиційні методи базуються на шаблонах (template matching). Цей підхід передбачає порівняння кожного сегментованого символу з базою зразків символів, які вже відомі системі. Кожен символ зіставляється із шаблонами за формою та розміром, і система вибирає найбільш відповідний варіант. Хоча цей метод є досить простим, він часто менш ефективний при роботі з різними шрифтами, стилями або спотвореннями символів.

З появою більш сучасних методів розпізнавання тексту, більшість сучасних OCR систем використовують машинне навчання, зокрема згорткові нейронні мережі (CNN) та рекурентні нейронні мережі (RNN). Ці підходи дозволяють системам навчатися на великій кількості різноманітних текстів і шрифтів, що значно підвищує їхню здатність розпізнавати символи навіть у складних умовах. Моделі CNN використовуються для аналізу форми та структури кожного символу, враховуючи його особливості, як-от контури та заповнені області.

Рекурентні нейронні мережі (RNN), зокрема Long Short-Term Memory (LSTM) (див. Рис. 1.5) моделі, часто використовуються для контекстуального аналізу тексту. Вони дозволяють враховувати послідовність символів і розпізнавати їх із урахуванням попередніх і наступних символів у тексті. Це корисно для корекції можливих помилок розпізнавання або для уточнення вибору символів у складних випадках.

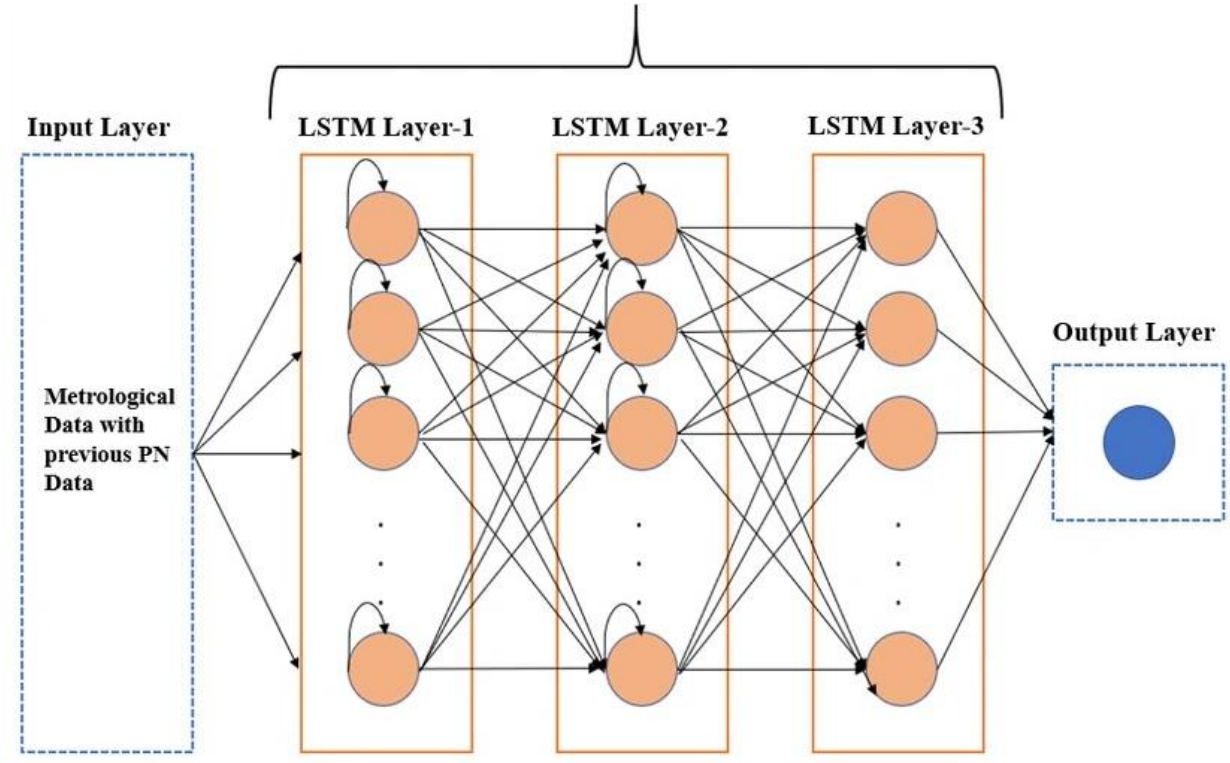

Рисунок 1.5 – Схема алгоритму Long Short-Term Memory

Алгоритми на основі відстані Левенштейна також можуть використовуватися для перевірки правильності розпізнаних символів. Цей

алгоритм вимірює, наскільки подібний розпізнаний символ до інших можливих варіантів, і допомагає системі вибрати найбільш відповідний символ або слово на основі контексту та ймовірності.

Після розпізнавання кожного символу система об'єднує їх у слова та речення. На цьому етапі можуть застосовуватися додаткові фільтри та алгоритми перевірки, такі як зіставлення зі словниками або мовними моделями, щоб виправити помилки або підвищити точність результату, допомагає виправляти неправильно розпізнані символи або покращувати граматичну точність тексту.

Тож процес розпізнавання символів включає в себе сегментацію, використання алгоритмів шаблонного зіставлення або машинного навчання для ідентифікації символів для забезпечення точності та коректності кінцевого результату [8].

Крок 4: Контекстний аналіз. Окрім розпізнавання окремих символів, алгоритми розпізнавання враховують контекст навколишніх символів і слів, щоб підвищити точність. Це може включати аналіз словосполучень, мовних моделей і граматичних правил, щоб вивести правильну інтерпретацію неоднозначних символів або слів.

Контекстний аналіз в OCR відіграє вирішальну роль у покращенні точності розпізнавання тексту, зокрема у виправленні помилок, пов'язаних з неправильно розпізнаними символами або словами. Він базується на використанні контекстної інформації, що дозволяє системі не просто розпізнавати окремі символи, а й враховувати взаємозв'язки між ними, що значно підвищує точність результату.

Контекстний аналіз починається з оцінки послідовності символів у тексті. Наприклад, якщо система помилково розпізнала окремі символи, вона може врахувати найбільш вірогідні варіанти слів, засновані на їх розташуванні у реченні та ймовірності появи певних слів разом. Це особливо корисно в ситуаціях, коли символи виглядають схожими, наприклад, коли букву "0" сплутали з "o" або цифру "1" – з літерою "l" (див. Рис. 1.6) [7].

Рисунок 1.6 – Типові помилки під час розпізнавання символів, схожих на інші

Одним із ключових методів контекстного аналізу є використання мовних моделей. Ці моделі, зазвичай побудовані на великих наборах даних, можуть передбачати правильні слова або фрази, виходячи з контексту. Наприклад, якщо OCR-система розпізнала слово неправильно, мовна модель може виправити це, базуючись на тому, які слова зазвичай вживаються в цьому контексті. Використання моделей на основі n-грамів або більш складних мовних моделей, таких як нейронні мережі, дає можливість аналізувати послідовність символів та слів і передбачати правильні варіанти розпізнавання.

Крім того, контекстний аналіз дозволяє враховувати граматичні та синтаксичні правила мови. Якщо виявлене слово не відповідає загальним правилам граматики чи структури мови, система може запропонувати його заміну на основі більш відповідного варіанту. Це зменшує ймовірність граматичних помилок, що можуть виникнути під час розпізнавання тексту.

Окрім мовних моделей, для контекстного аналізу можуть використовуватися також алгоритми машинного навчання та нейронні мережі.

Вони здатні аналізувати текст на більш високому рівні, враховуючи не лише окремі символи або слова, але й більш складні взаємозв'язки між ними, такі як лексичні патерни або фразеологічні звороти. Це особливо корисно в текстах, де використовуються специфічні терміни або абревіатури, які можуть бути неправильно розпізнані без урахування контексту.

У деяких випадках контекстний аналіз також включає роботу зі спеціалізованими словниками або базами даних для певних галузей. Наприклад, у медичних або технічних текстах можуть використовуватися специфічні терміни, які стандартні мовні моделі можуть не враховувати. У таких випадках застосовуються спеціалізовані алгоритми, які покращують розпізнавання тексту в межах конкретного контексту.

Останнім етапом контекстного аналізу є перевірка логіки та сенсу тексту. Це може включати аналіз стилю та тону тексту, що дозволяє системі виявити будь-які невідповідності або помилки в структурі речень. Наприклад, якщо слово виглядає незрозумілим або не відповідає загальному змісту, система може звернути увагу на контекст і запропонувати варіанти його заміни [7, 8].

Крок 5: Постобробка. Після розпізнавання символів програмне забезпечення OCR виконує завдання постобробки для уточнення результатів і виправлення помилок. Це може включати перевірку орфографії (див. Рис. 1.7), алгоритми виправлення помилок і оцінку достовірності для виявлення та виправлення неточностей у розпізнаному тексті.

Рисунок 1.7 – Постобробка для розпізнавання символів на прикладі граматики

Постобробка в OCR – це завершальний етап обробки тексту після розпізнавання символів, який забезпечує підвищення точності результату та коригування можливих помилок. Основна мета постобробки – виправити неправильно розпізнані символи, покращити структуру тексту, а також зробити його більш зрозумілим і відповідним до реального тексту на зображенні.

Процес постобробки починається з перевірки граматичної та лексичної правильності розпізнаного тексту. Система порівнює кожне розпізнане слово зі словником відповідної мови. Якщо слово не знайдено в базі даних словника, воно вважається помилковим, і система намагається знайти можливі варіанти виправлення на основі схожості з іншими словами або контексту.

Алгоритми перевірки правопису є одним із головних інструментів на цьому етапі. Вони застосовують статистичні методи або машинне навчання для передбачення правильного варіанту слова. Наприклад, можуть використовуватися методи на основі відстані Левенштейна, які вимірюють,

наскільки схоже слово з іншими можливими варіантами, і допомагають вибрати найбільш ймовірне правильне слово. Цей підхід дозволяє виправляти типові OCR-помилки, такі як плутанина між схожими літерами (наприклад, "o" і "0", "l" і "1").

Крім правопису, важливою частиною постобробки є корекція структури тексту. OCR системи можуть неправильно інтерпретувати межі слів або пропуски між символами, особливо якщо текст був спотворений або має незвичайні інтервали. Постобробка включає корекцію цих проблем, щоб відновити правильні розриви між словами і реченнями, що підвищує читабельність і точність тексту [7].

Контекстуальний аналіз також відіграє важливу роль у постобробці. Багато систем OCR використовують моделі мовних мереж або алгоритми машинного навчання для аналізу контексту розпізнаних слів і фраз. Це дозволяє виправляти помилки в залежності від того, які слова зустрічаються поруч. Наприклад, якщо неправильно розпізнане слово виглядає незрозумілим в контексті речення, система може запропонувати інші варіанти виправлення.

У деяких випадках, особливо коли текст містить спеціальні символи, технічні терміни або специфічні абревіатури, можуть використовуватися спеціальні доменні словники. Це дозволяє точніше розпізнавати текст у вузькоспеціалізованих областях, таких як медицина, юридичні документи або наукові праці, де звичайні словники можуть не бути ефективними.

Ще одним важливим аспектом постобробки є відновлення форматування тексту. Під час OCR часто втрачаються оригінальні стилі, такі як жирний шрифт, курсив або підкреслення. Деякі сучасні OCR системи можуть спробувати відновити форматування на основі аналізу оригінального зображення, хоча це часто потребує додаткових інструментів для обробки документів.

Останнім етапом є перевірка логічної та смислової послідовності тексту. Цей процес включає в себе аналіз структури документу, наприклад, корекцію

заголовків, абзаців, нумерації сторінок, а також відновлення таблиць і графіків, якщо вони були присутні в оригінальному документі [7].

Крок 6: Форматування вихідних даних. Нарешті, розпізнаний текст виводиться в машинозчитуваному форматі, наприклад, у звичайному тексті, розширеному текстовому форматі (RTF) або PDF з можливістю пошуку (див. Рис. 1.8). Програмне забезпечення розпізнавання може також зберігати оригінальну структуру та форматування документа, зокрема шрифти, стилі та елементи форматування, щоб зберегти візуальну цілісність тексту.

Рисунок 1.8 – Вихідні дані розпізнавання символів

Форматування вихідних даних в OCR – це завершальний етап обробки, який полягає у відновленні структури та вигляду тексту відповідно до оригінального документа. Після того як текст розпізнано і оброблено, система повинна зберегти або відновити форматування, яке було присутнє в початковому зображенні. Це робить кінцевий результат більш точним і зручним для подальшого використання.

Процес форматування вихідних даних починається з аналізу макета документа. OCR системи визначають розташування різних елементів на сторінці, таких як абзаци, заголовки, підзаголовки, списки, таблиці, зображення та інші графічні елементи. Після цього система намагається зберегти відносні позиції цих елементів, щоб відновити оригінальний вигляд

документа. Це особливо важливо для складних документів, які містять не лише текст, але й елементи форматування, такі як таблиці або стовпці.

Одним із найважливіших аспектів форматування є збереження правильних відступів і структури абзаців. Під час процесу розпізнавання тексту OCR може втратити ці елементи, особливо якщо зображення містить нерівномірні відступи або різні стилі шрифтів. Постобробка дозволяє відновити ці параметри на основі аналізу оригінального документа, забезпечуючи правильну розбивку тексту на абзаци, заголовки та списки. Це значно покращує читабельність кінцевого тексту.

Збереження шрифтів і стилів є ще одним важливим аспектом форматування вихідних даних. Оригінальні документи можуть містити текст у різних шрифтах, розмірах і стилях (жирний, курсив, підкреслений). У процесі OCR система намагається розпізнати ці стилі і зберегти їх у кінцевому документі. Для цього використовуються алгоритми, які аналізують візуальні характеристики тексту на зображенні. Наприклад, система може розпізнати, що певна частина тексту виділена жирним шрифтом або курсивом, і застосувати це форматування до кінцевого результату.

Особливу увагу приділяють відновленню таблиць та інших структурованих даних. Таблиці є важливим елементом багатьох документів, але їх розпізнавання може бути складним завданням для OCR систем через можливі спотворення або нерівні лінії. Після розпізнавання таблиць система намагається відновити їх структуру, включаючи кількість стовпців і рядків, а також вирівнювання тексту в кожній комірці. Це забезпечує точне відтворення оригінальних даних, зберігаючи їхню структуру для подальшого аналізу або використання.

Для документів, які містять списки, відновлення маркерів або нумерації також є частиною форматування вихідних даних. OCR система повинна правильно розпізнати тип списку (нумерований чи маркерний) і відновити його у вихідному тексті. Це дозволяє зберегти структуру списків і зробити документ легшим для сприйняття.

Крім того, форматування може включати відновлення відстаней між рядками, вирівнювання тексту та інших елементів макета. Наприклад, система може відновити вирівнювання тексту (ліворуч, праворуч, по центру або по ширині), що допомагає створити візуально привабливий і логічно структурований документ.

Для багатомовних документів форматування також може включати правильне розташування різних мовних елементів, особливо якщо вони мають різні стилі або шрифти. У таких випадках OCR системи повинні коректно обробити всі частини тексту, щоб уникнути змішування форматування або некоректного розташування символів.

Останнім етапом форматування є експорт кінцевих даних у потрібний формат. Багато OCR систем дозволяють користувачам зберігати розпізнаний текст у різних форматах, таких як PDF, Word, Excel або HTML. Кожен із цих форматів має свої вимоги до форматування, тому система повинна забезпечити відповідність кінцевого результату стандартам обраного формату. Наприклад, при збереженні документа у форматі PDF система зберігає макет сторінки, включаючи зображення, таблиці та інші графічні елементи [8].

Машинне навчання дозволяє системам розпізнавати різноманітні шрифти, макети та мови, підвищуючи точність та універсальність. Алгоритми вправно справляються зі складними структурами та варіаціями документів, що знаменує собою значний стрибок в ефективності обробки документів.

Крім того, машинне навчання дає можливість OCR вчитися на великих масивах даних, вдосконалювати свої можливості розпізнавання та зменшувати кількість помилок. Сучасні системи розпізнавання використовують нейронні мережі, що підвищує здатність технології підтримувати високу точність розпізнавання різних типів документів. Наприклад:

1. Розпізнавання друкованого тексту. Розпізнавання друкованого тексту розпізнає і витягує текст з документів зі стандартними друкованими шрифтами. Цей тип оцифровує друковані матеріали, такі як книги, статті або

офіційні документи, з високою точністю. Незвичайні макети документів, погане форматування та складний фон можуть створювати проблеми під час розпізнавання друкованого тексту.

2. Розпізнавання рукописного тексту. Розпізнавання рукописного тексту перетворює рукописний текст у машинозчитувані символи. Це складне завдання через різноманітність стилів почерку. Точне розпізнавання рукописного тексту вимагає вдосконалених моделей машинного навчання, навчених на різноманітних наборах даних рукописних зразків.

3. Розпізнавання сюжетного тексту. Розпізнавання сюжетного тексту спеціалізується на вилученні тексту із зображень, захоплених у реальних сценах, таких як вуличні знаки, етикетки товарів або шаблони плакатів. Аналіз тексту в динамічних сценах зі змінним освітленням і спотвореннями вимагає просунутого комп'ютерного зору та алгоритмів розпізнавання тексту.

Переваги оптичного розпізнавання символів включають в себе:

1. Підвищення ефективності та продуктивності за допомогою OCR

Технологія оптичного розпізнавання символів підвищує продуктивність, автоматизуючи завдання введення даних. Вона усуває необхідність ручного введення інформації з паперових документів у цифрові системи шляхом автоматичного вилучення тексту зі сканованих зображень.

2. Покращена доступність для людей з вадами зору завдяки розпізнаванню тексту.

OCR перетворює друкований або рукописний текст у цифровий формат. Це дозволяє програмному забезпеченню для перетворення тексту в мовлення читати його вголос, що значно полегшує людям з вадами зору доступ до письмової інформації та її розуміння.

3. Покращений пошук та організація документів за допомогою OCR.

OCR перетворює статичні документи на динамічні файли з можливістю пошуку. Перетворення зображень на машинозчитуваний текст забезпечує швидкий та ефективний пошук у документах. Користувачі можуть швидко

знаходити потрібну інформацію у великих обсягах документів, що робить управління документами більш ефективним і зручним для користувачів.

4. Зменшення залежності від паперу та місця для зберігання.

Оцифровуючи документи, OCR зменшує витрати, пов'язані з друком, зберіганням та управлінням паперовими копіями. Це екологічно безпечно, економить простір і створює більш впорядковане та стійке робоче середовище;

5. Збереження та оцифрування історичних документів.

Розпізнавання тексту (OCR) має вирішальне значення для збереження історичних документів шляхом оцифрування старовинних рукописів, делікатних записів і старих газет. Це захищає цінний історичний контент від псування і робить його доступним для ширшої аудиторії [9].

Галузеві випадки використання оптичного розпізнавання символів:

1. Автоматизація бізнес-процесів: OCR використовує технологію для оцифрування даних із зображень і PDF-файлів. Раніше компаніям доводилося вирішувати трудомісткі завдання вручну, що призводило до помилок і неефективності. OCR вирішує цю проблему, швидко перетворюючи паперові документи в цифрові дані. Рахунки-фактури та квитанції можна легко обробляти за допомогою OCR, що прискорює робочі процеси та мінімізує ризик помилок.

2. Охорона здоров'я: OCR допомагає в управлінні інформацією в секторі охорони здоров'я. Медичні працівники можуть миттєво отримати доступ до інформації про пацієнтів, оцифрувавши їхні історії хвороби та рецепти. Це забезпечує краще ведення записів і швидкий пошук історії хвороби.

3. Освіта: розпізнавання тексту в освіті спрощує такі завдання, як сканування підручників і конспектів лекцій. У минулому студенти та викладачі стикалися з проблемами при роботі з великими обсягами друкованих матеріалів.

Розпізнавання тексту стало рятівником, перетворивши ці матеріали в цифрові формати, створивши динамічне та інтерактивне навчальне

середовище. Тепер студенти можуть легко шукати, виділяти та обмінюватися інформацією, що сприяє кращому навчанню.

4. Юридична сфера: система OCR підвищує доступність і спрощує зберігання юридичних даних, роблячи файли більш динамічними та зручними для використання. OCR спрощує процеси, перетворюючи паперові документи на цифрові файли з можливістю пошуку.

Тепер юристи можуть оптимізувати управління справами, підвищити ефективність досліджень і створювати впорядковані бібліотеки правової інформації. Легкий і швидкий пошук інформації також покращує швидкість судових процесів.

5. Подорожі та туризм: у туристичному секторі OCR прискорює вилучення інформації з документів, пов'язаних з подорожами, таких як посадочні талони, паспорти, візи та квитки. Авіакомпанії, імміграційні служби та інші туристичні організації використовують OCR для прискорення процесів реєстрації на рейси, скорочення часу очікування та покращення якості обслуговування мандрівників.

6. Засоби масової інформації та видавнича справа: оцифровуючи архіви та газети за допомогою OCR, медіа-організації успішно підвищують довговічність цінних документів.

OCR надає журналістам швидкий доступ до історичних даних і дозволяє перепрофілювати контент для ширшої аудиторії. Це зробило історичні документи легкодоступними в цифровому просторі, сприяючи таким чином збереженню джерел історичних знань [10, 11].

1.3. Класифікація систем розпізнавання символів

В основному, методи розпізнавання символів включають в себе отримання інформації із зображення і впровадження її в систему розпізнавання. Розпізнавання символів – це дуже велика область, яку абстраговано можна класифікувати таким чином:

1. Розпізнавання символів в режимі он-лайн.

2. Розпізнавання символів в автономному режимі.

На основі даних, отриманих в результаті вищеописаного процесу, системи можна розділити на категорії, як показано за допомогою Рис. 1.9.

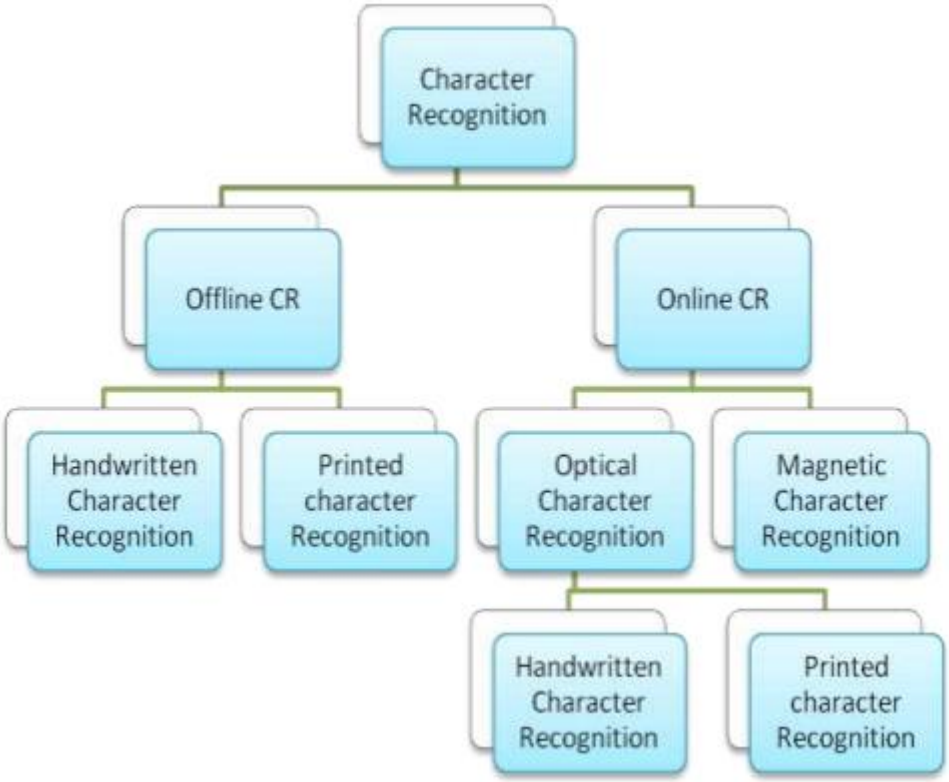

Рисунок 1.9 – Схема класифікації систем розпізнавання символів

Розпізнавання символів онлайн – це процедура сприйняття символів так, як вони написані в контенті. Крім того, фреймворк он-лайн розпізнавання символів має безперервні тести розмовної інформації фреймворків, які використовують он-лайн розпізнавання, що охоплює Apple Newton, Palm Pilot, мобільні телефони з бітовим екраном. Якщо виникає необхідність розпізнавання складених символів в режимі он-лайн, складена інформація з зображення сприймається і ретельно структурується, далі ця вдосконалена структура пропускається через фреймворк розпізнавання символів, щоб зібрана інформація могла бути сприйнята. Загалом, зображення поміщається у фреймворк і після цього готується.

З іншого боку, офлайн-розпізнавання може бути системою, яка розпізнає шляхом захоплення зображення символів або письмового тексту, які мають відповідну площу для розпізнавання. Потенціал офлайн-систем розпізнавання лежить у таких сферах, як документообіг, пересилання пошти та перевірка чеків.

Офлайн-розпізнавання почерку означає розпізнавання слів, які досліджуються, з паперового аркуша і площі та збереження їх у вигляді відтінків сірого. Як тільки це зроблено, це стає викликом для інших додаткових процесів для дозволу попередніх процесів, однак офлайн-знання не підтримують інформацію про дискурс у реальному часі. Ця відмінність породжує проблему відмінностей у процесах, які необхідно виконати.

Розпізнавання символів (офлайн) може диверсифікуватися на дві додаткові категорії

1. Магнітне розпізнавання символів (MCR).
2. Оптичне розпізнавання символів (OCR) [12].

Магнітне розпізнавання символів (MCR) (див. Рис. 1.10): кожен символ написаний чорнилом, тому в MCR пристрій для зчитування розпізнає кожен символ як унікальний символ і силу кожного символу. MCR зазвичай використовується в банках для перевірки служби автентифікації та додатково для внесення змін до групових заяв на проведення операцій.

Рисунок 1.10 – Демонстрація магнітного розпізнавання символів

Оптичне розпізнавання символів (OCR) (див. Рис. 1.11): відоме як ідентифікація символів, отриманих шляхом сканування зображення або фотографування.

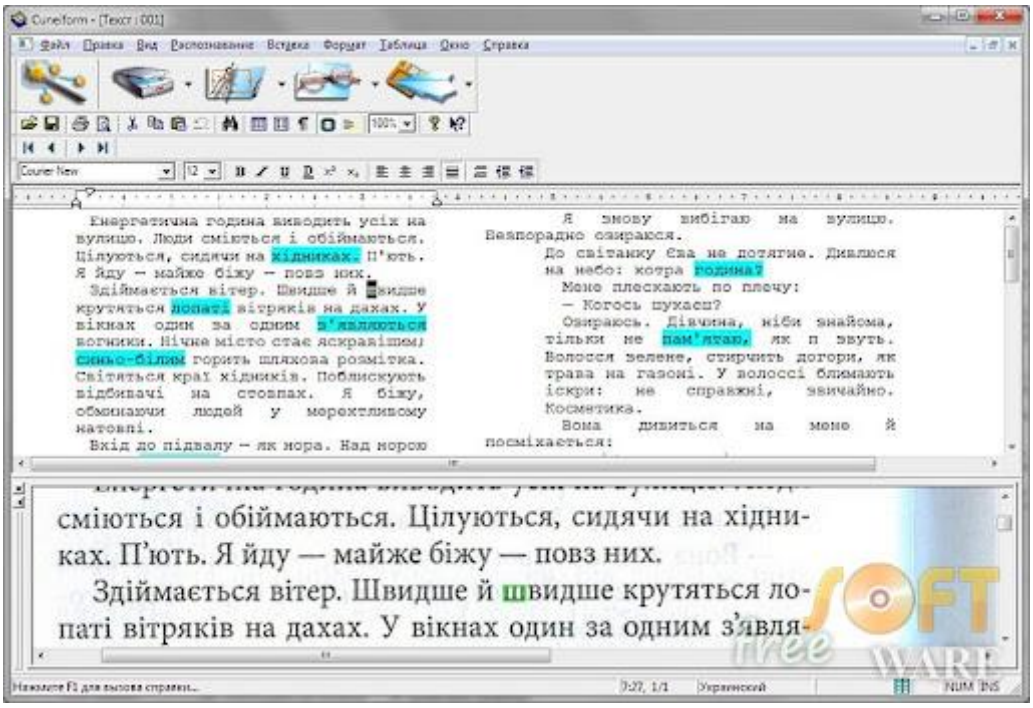

Рисунок 1.11 – Демонстрація оптичного розпізнавання символів

Символи, як правило, являють собою зображення, розділені на частини, написані від руки або надруковані, різного розміру і різної форми. OCR поділяється на розпізнавання друкованих і рукописних символів.

Розпізнавання символів рукописних сторінок складніше, ніж друкованих символів, що пов'язано з різною структурою почерку. Розпізнавання друкованих символів дещо простіше, оскільки зображення мають приблизно однаковий формат [13].

1.4. Необхідний набір даних для розпізнавання тексту, перевірка ASCII-відповідності зображень

Для того, щоб повністю зрозуміти вплив помилок розпізнавання на ефективність пошуку, необхідно створити колекцію документів, яка буде

гарним тестовим набором як для розпізнавання, так і для IR-досліджень (information retrieval).

На жаль, через брак робіт з інтеграції цих двох технологій не існує баз даних, які б вирішували питання, пов'язані з аналізом документів та IR. З точки зору аналізу документів, колекція повинна складатися з документів з численними шрифтами, різною якістю паперу, містити різноманітні графічні матеріали (наприклад, формули, графіки, фотографії, карти). З точки зору IR, колекція повинна охоплювати різноманітні теми, мати різний обсяг документів, різні стилі написання, а також набір запитів і відповідних суджень про релевантність.

Крім того, для проведення досліджень, пов'язаних з помилками розпізнавання, ми повинні мати колекцію, розпізнану OCR, і відповідну колекцію, виправлену вручну. Різні цілі при створенні колекцій для комбінованих експериментів ускладнюють їхню побудову. Перевірка точності розпізнавання за допомогою мільйона символів (приблизно 500 сторінок) вважається значною, і це справедливо. Підготовка базової сторінки для розпізнавання вимагає нудного передруку, синхронізації та перевірки з точністю до 100%.

IR-тестування, для порівняння, вимагає набагато більших тестових колекцій, щоб зробити експериментальні результати значущими.

Існує дві альтернативи: створення колекції вручну та моделювання помилок розпізнавання за стандартними колекціями. Наші експерименти включають обидва ці методи. Моделювання проводилося шляхом деградації чотирьох стандартних колекцій IR-зображень.

Колекція, створена вручну науковцями Kazem Taghva, Julie Borsack, Allen Condit, складається з 674 документів (26 467 сторінок) і є частиною великої колекції, яку було створено підрядниками для Міністерства енергетики США (DOE) разом із прототипом Системи підтримки ліцензування (LSS). LSS збиратиме та відстежуватиме документи, що

стосуються процедур ліцензування ядерних об'єктів Комісією ядерного регулювання [14].

Як і слід було очікувати від такої колекції, як LSS, більшість документів стосуються технічних, наукових тем. Хоча документи тяжіють до наукової тематики, в межах цієї сфери колекція є різноманітною. Наш набір охоплює теми від видобутку корисних копалин до питань безпеки при транспортуванні ядерних відходів. Вона охоплює всі 16 встановлених предметних областей, визначених для LSS.

Характеристики цієї колекції документів, які роблять її особливо корисною для демонстрації наслідків використання даних розпізнавання текстів, полягають у відсутності однаковості формату сторінок, різноманітності стилів шрифтів і різному рівні якості друкованої продукції. Існує майже стільки ж різних авторських джерел, скільки й документів. Отже, хоча колекція є переважно науковою, широкий спектр джерел дає нам багату колекцію для тестування.

Колекція складається з повнотекстових документів, середній обсяг яких становить 40 сторінок, а медіана – 16 сторінок. Зображення були відскановані на сканері Fujitsu Image Scanner, номер моделі M3096G, з роздільною здатністю 300 dpi (dots per inch – кількість точок на дюйм) (див. Рис. 1.12) з медіанним порогом 127 [15].

Рисунок 1.12 – Порівняння значень dpi

Невідомо, які процедури сканування були застосовані до зображень, створених державними підрядниками для ОУЖД. Ми знаємо, що ці зображення були відскановані сканерами Ricoh або Fujitsu з роздільною здатністю 300 dpi. Усі 674 документи були розпізнані трьома різними пристроями розпізнавання тексту для нашого експерименту.

Критерієм відбору цих пристроїв був рівень точності розпізнавання символів. Три пристрої, які ми обрали, мали найвищу точність (98,14%), середню точність (97,06%) і найнижчу точність розпізнавання символів (94,63%) серед пристроїв, протестованих в ISRI. Для простоти використовуємо такі позначення для трьох пристроїв відповідно: найкращий, середній і найгірший. Ці позначення стосуються лише точності розпізнавання символів – переваги інших характеристик пристрою (наприклад, простота використання, пропускна здатність) не бралися до уваги. Четверта база даних, позначена як правильна, складається з 70 виправлених ASCII для цих 674 документів.

Мета експериментів – з'ясувати, які наслідки матимуть дані OCR при використанні в IR-системах. Для того, щоб зробити таку оцінку, ми повинні спочатку знати характеристики не тільки тексту, створеного OCR, але й виправленого тексту, з яким його порівнюємо. Аналіз наших баз даних став значною частиною нашого експерименту. Наш аналіз складається з двох частин: Перевірка документів у форматі ASCII/OCR та статистичне дослідження колекцій IR-систем. На момент написання цієї статті колекція LSS налічувала 1349 документів, для яких усі компоненти, окрім суджень про релевантність, були повними.

До експериментальних наборів додавалася все більша кількість учасників у міру того, як завершувалося опрацювання більшої кількості одиниць документів (зображення, текст OCR, судження про релевантність). Стандартні колекції IR використовувалися лише для імітаційних експериментів.

Відповідність ASCII-документа його оригінальній паперовій версії, а отже, і його зображенню, була важливою для тестування. З першої серії експериментів з'ясувано, що ця відповідність не завжди може бути такою, як ми очікуємо. Тому для кожного документа в нашій текстовій базі даних ми порівняли кожну сторінку паперової версії, її зображення та ASCII-сторінку. Відповідність між друкованою сторінкою і сторінкою зображення легко виявити, оскільки зображення є електронною копією друкованої сторінки.

Про відповідність між друкованою версією та її ASCII-еквівалентом судити складніше. Певні якості та компоненти документа не можуть бути продубльовані в його ASCII-версії, наприклад, збільшений відбиток або фотографія. Але в якийсь момент необхідно встановити правила включення або виключення елементів документа. Ці правила були встановлені для виправленої версії ASCII, створеної для LSS. На жаль, з чотирма різними підрядниками ці правила були залишені для інтерпретації. Крім того, щоразу, коли в проекти такого масштабу включається людський фактор, результати важко передбачити. Під час порівняння виявлено деякі несподівані відмінності. Найсуттєвішою відмінністю, що вплинула на наші експерименти, було виключення тексту з правильних ASCII-файлів. У деяких випадках це виключення було цілеспрямованим, а в інших – через недбалість. Наприклад, одним із правил редагування для LSS була заміна таблиць і графіки тегом [16].

У ряді випадків ці елементи містили значну частину тексту. Серед інших невідповідностей – додавання тексту, який не був частиною оригінального документа, помилкова колонцифровка, неправильний порядок сторінок і видалення основного тексту. У кількох випадках документи довелося виключити з нашого тестування через їхню недостатню схожість з оригінальною друкованою версією. Хоча в більшості випадків виправлений ASCII є досить якісним, виявлені нами відмінності свідчать про те, що при втручанні людини, навіть з метою виправлення, слід очікувати певних помилок і невідповідностей. Припускається, що проблеми, подібні до цих,

притаманні будь-якій процедурі ручного виправлення і тому є невід'ємною частиною нашого експерименту.

Перевірка ASCII встановлює корисність виправленої колекції документів. Ми також повинні дослідити якості наших колекцій OCR, щоб зрозуміти, як вони впливають на наші результати. Загалом, шукали грубі відмінності між зображенням і згенерованим текстом. Наприклад, якщо на сторінці, згенерованій OCR, були блоки безглуздих символів, це могло вказувати на можливі проблеми із зображенням. Ці порушення було перевірено для всіх сторінок, створених за допомогою OCR, використаних у нашій колекції.

Різниця, виявлена серед колекцій OCR, полягала у кількості сторінок, які могли бути розпізнані кожним пристроєм. Із загальної кількості 26 467 зображень сторінок, розпізнані одним пристроєм сторінки не обов'язково були підмножиною сторінок, розпізнаних іншими. Ще однією відмінністю між пристроями є їхня здатність розпізнавати та «виокремлювати» графіку. Якщо пристрій розпізнавання тексту має функцію автоматичного зонування, він повинен розрізняти текстові та нетекстові частини (тобто фотографії, карти тощо); він повинен перекладати тільки ті, які були позначені як текстові «зони».

Під час тестування виявлено, що таке вміння може бути важливим для деяких інфрачервоних систем. Відсутність точного зонування не лише збільшує накладні витрати, але й може обмежити ефективність IR-перекладу. Усі пристрої забезпечували якісний вивід із якісних зображень сторінок. Розбіжності в точності виводу стали більш очевидними серед трьох пристроїв, коли якість зображень знизилася [17].

1.5. Алгоритми розпізнавання тексту на основі машинного навчання

Машинне навчання – це дисципліна, яка вивчає, як змусити комп'ютерні програми імітувати людський людський мозок для навчання. Він вимагає, щоб

комп'ютер аналізував дані про навколишнє середовище, узагальнював їх і створював відповідну базу знань самостійно, подібно до людського мозку. відповідну базу знань самостійно, подібно до людського мозку, щоб покращити продуктивність алгоритму на основі результатів навчання.

Машинне навчання може не тільки самонавчатися, щоб поліпшити виконання алгоритму, але й узагальнювати велику кількість даних, що дає машинному машинному навчанню значну перевагу у вирішенні великомасштабних задач класифікації та розпізнавання з нечіткими внутрішніми характеристиками зображення.

На Рис. 1.13 – модель машинного навчання для розпізнавання тексту, запропонована Саймоном, одним із провідних фахівців у галузі штучного інтелекту.

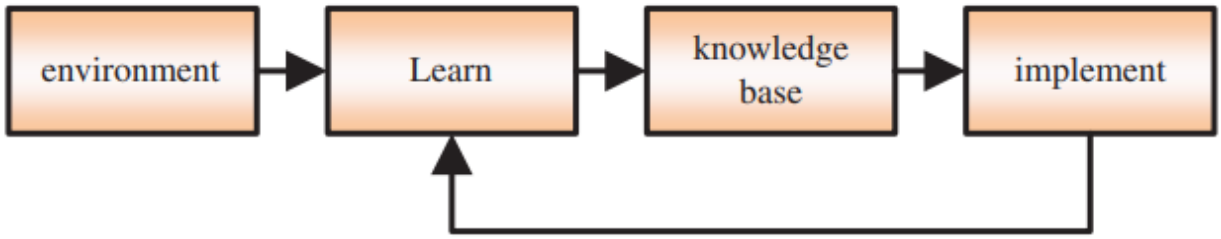

Рисунок 1.13 – Модель машинного навчання для розпізнавання тексту

Процес машинного навчання виглядає наступним чином: навколишнє середовище надає вхідні дані навчальній системі, а потім алгоритм навчання всередині навчальної системи отримує дані, аналізує дані та перетворює їх у знання; база знань отримує та зберігає знання, а дані в базі знань впливають на подальше введення даних; після того, як виконавець виконає завдання, він повертає результат навчальній системі, а потім навчальна система аналізує результат та реагує; процес продовжується до тих пір, поки не буде виконано завдання навчання заданої мети, тобто навчальна система не завершить своє навчання [18].

У навчальній системі для того, щоб сформулювати знання, необхідно дотримуватися наступних принципів:

1. Невелика обчислювальна складність. У процесі навчання учневі доводиться перетворювати велику кількість даних про навколишнє середовище на знання. Якщо метод перетворення занадто складний і вимагає великої кількості обчислень, це займе багато часу і вплине на загальну продуктивність алгоритму.

Наприклад, коли необхідно порівняти два вирази, щоб побачити, чи є вони еквівалентними, якщо вони є векторами ознак, судження можна зробити за допомогою швидкого і простого розрахунку відстані між двома векторами. тоді як, якщо вони на мові логіки першого порядку, необхідно перетворити два реляційні вирази до однакової форми виразу, перш ніж порівнювати їх, і цей процес збільшить кількість необхідних обчислень.

2. Легкість пошуку та модифікації. Навчальна система повинна постійно оновлювати та модифікувати зміст знань у базі знань у процесі навчання, тому знання в навчальній системі повинні мати чітку структуру та організацію, наприклад, правила зберігання та вектори ознак. У процесі оновлення бази знань, якщо нові необхідні знання ідентичні або суперечать старим, старі знання повинні бути видалені. Таким чином, в процесі розширення бази знань, існуючий зміст знань повинен бути перевірений і скоригований.

3. Розширювана база знань. Основна вимога до бази знань полягає в тому, що вона повинна виражати різні види знань. і ще одна вимога полягає в тому, щоб база могла будувати нові знання на основі існуючих знань, щоб вона могла розширювати поле знань і включати більш репрезентативні і більш складні дані, щоб поліпшити здатність системи навчання до виконання. Розширюваність бази знань є дуже важливим фактором, що впливає на продуктивність великомасштабних систем розпізнавання. Сучасні зрілі алгоритми ML включають наївний байєсівський алгоритм, алгоритм логістичної регресії, аналогічне навчання, лінійну регресію, алгоритм навчання нейронних мереж, SVM, алгоритм найближчого сусіда тощо.

Ці алгоритми можуть узагальнювати знання та розширювати базу знань, знаходячи правила між вхідними та вихідними даними вибірки для заданого розміру вибірки даних. Така структурна модель, яка може навчатися, генерувати певний зміст знань і класифікувати їх, відома як класифікатор. Завдяки моделям ML дослідникам більше не потрібно витрачати багато часу на аналіз даних і визначення правил для величезних баз даних. Замість цього вони можуть визначити, як виділити більш репрезентативні ознаки, і розробити класифікатори та моделі знань з кращою продуктивністю, а також забезпечити швидку збіжність класифікатора.

Саме завдяки цій перевазі ML здатен замінити традиційний пошук за шаблоном і, таким чином, відігравати важливу роль як у промисловості, так і в суспільстві.

Методи розпізнавання символів на основі статистичних класифікаторів в основному включають методи розпізнавання символів на основі SVM. SVM – це метод машинного навчання. Завдяки своїм чудовим показникам навчання, ця технологія швидко стала центром досліджень в галузі машинного навчання. SVM має великі переваги у вирішенні завдань з невеликою вибіркою.

Метод розпізнавання символів на основі SVM поділяється на етап навчання та етап тестування. Етап навчання полягає у вилученні ознак із зразків навчальної вибірки, встановленні параметрів SVM та навчанні SVM-класифікатора; етап тестування полягає у вилученні ознак із зразків тестової вибірки, підстановці ознак у навчений SVM-класифікатор, обчисленні значення рішення та оцінці категорії зразка на основі значення рішення. В роботі запропоновано алгоритм розпізнавання символів, що поєднує алгоритм KNN (K-Nearest Neighbor) та вдосконалений SVM [18].

Коли цей алгоритм використовується для тестування 1000-символьних зразків, швидкість розпізнавання становить 96,1%. Крім того, в роботі порівнюються показники розпізнавання декількох поширених методів розпізнавання символів для тих самих зразків з 1000 символів. Серед них, швидкість розпізнавання методу зіставлення шаблонів на основі відстані

Хаусдорфа склала 93,5%, а методу на основі нейронної мережі BP – 93,2%. Результати експерименту показали, що швидкість розпізнавання алгоритму розпізнавання символів, що поєднує KNN та покращену SVM, є вищою, ніж у методу зіставлення шаблонів на основі відстані Хаусдорфа та методу на основі нейронної мережі BP. У нелінійній моделі SVM загальні функції ядра такі:

Лінійна функція kernel:

$$K(x_1, x_2) = (x_1, x_2),$$

де x_1, x_2 – аргументи функції.

Поліноміальна функція kernel:

$$K(x_1, x_2) = ((x_1, x_2) + R)^d,$$

розмірність цього простору дорівнює C^d_{p+d}, де p – початкова просторова розмірність.

Гаусова функція kernel:

$$K(x_1, x_2) = \exp\left(\frac{-||x_1 - x_2||^2}{2\sigma^2}\right)$$

Коли σ занадто велике, вага атрибутів ознак високого порядку швидко зменшується, а значення все ще еквівалентне низьковимірному простору; коли σ занадто мале, будь-які дані можуть бути лінійно відокремлюваними і схильними до надмірної підгонки.

Сигмоїдна функція kernel:

$$K(x_1, x_2) = \tanh(\gamma(x_1, x_2) + c),$$

де γ і c – константи.

Переваги алгоритму SVM полягають у наступному:

1) Він може бути використаний для лінійної/нелінійної класифікації та регресії, з низьким рівнем помилки узагальнення, що означає, що він має

хорошу здатність до навчання, а отримані результати можуть бути узагальнені.

2) Він може вирішити проблему машинного нахилу у випадку невеликих вибірок та проблем високої розмірності, а також уникнути проблем вибору структури нейронної мережі та локальних мінімумів.

3) SVM є найкращим готовим класифікатором. «Готовий» означає, що його можна використовувати безпосередньо без модифікації. Крім того, SVM має нижчий рівень помилок, і може приймати хороші класифікаційні рішення для точок даних поза навчальною вибіркою.

Як алгоритми ML, SVM мають наступні недоліки:

1) SVM дуже добре працює при обробці невеликих вибірок, оскільки його здатність до узагальнення є найвищою серед алгоритмів класифікації. Однак, коли ми стикаємося з великомасштабними даними високої розмірності, його продуктивність обмежена.

2) Він чутливий до пропущених даних.

3) Ядро машини опорних векторів є дуже потужною моделлю, яка добре працює на різних наборах даних. SVM дозволяє визначити складну границю рішення, навіть якщо є лише кілька ознак даних. Вона добре працює як на даних низької розмірності, так і на даних високої розмірності (тобто, мало ознак і багато ознак), але не дуже добре працює при масштабуванні кількості вибірок. Він може добре працювати на даних, що містять до 10 000 вибірок, але якщо кількість вибірок наближається або перевищує 100 000, можуть виникнути проблеми з часом виконання та використанням пам'яті.

4) SVM вимагає ретельної попередньої обробки даних та налаштування параметрів.

Ось чому в багатьох додатках сьогодні алгоритми використовують деревовидні моделі, такі як випадкові ліси та градієнтно-підсилені дерева (які потребують незначної попередньої обробки або взагалі не потребують її). Крім того, його процеси та методи прогнозування настільки складні, що їх опис неможливо донести до неспеціалізованої аудиторії [18].

1.6. Корекція слів для постобробки OCR

Архітектура системи корекції слів для постобробки OCR є важливим елементом, що дозволяє підвищити точність розпізнавання тексту та забезпечити більш високу якість кінцевого результату. Оптичне розпізнавання символів (OCR) часто генерує помилки через різні фактори, такі як низька якість зображень, неправильне освітлення, або спотворення документів. Система корекції слів розробляється з метою виправлення цих помилок на етапі постобробки.

Основними компонентами такої системи є модуль для попереднього аналізу тексту, база даних лексичних елементів та модуль алгоритмів корекції. Перший етап роботи системи полягає у проведенні аналізу отриманого тексту, зокрема виявленні слів, які не відповідають жодному з відомих мовних патернів або які ймовірно містять помилки. Для цього застосовуються різні методи лінгвістичного аналізу, зокрема зіставлення слів з наявними у базі даних лексемами, аналіз контексту та використання регулярних виразів.

Однією з ключових складових системи є база даних лексичних елементів, яка містить велику кількість слів, фраз і мовних конструкцій. Ця база даних слугує джерелом для верифікації правильності слів, а також для пошуку можливих варіантів заміни у разі виявлення помилки. Сучасні системи використовують адаптивні моделі, які враховують різні стилі мови, жаргон або специфічну термінологію, що дозволяє коригувати текст з урахуванням контексту [19].

Алгоритми корекції є основним інструментом для виправлення помилок. Одним з найбільш поширених підходів є використання методів машинного навчання та статистичних моделей для передбачення ймовірних варіантів корекції. Окрім цього, широко використовуються алгоритми на основі відстані Левенштейна для визначення подібності між неправильним та потенційно правильним словом.

Тобто система корекції слів для постобробки OCR є багатокомпонентною структурою, яка забезпечує виправлення помилок шляхом лінгвістичного аналізу, використання баз даних лексем та застосування алгоритмів машинного навчання й статистичних моделей.

Архітектура системи корекції слів для постобробки OCR показана на Рис. 1.14.

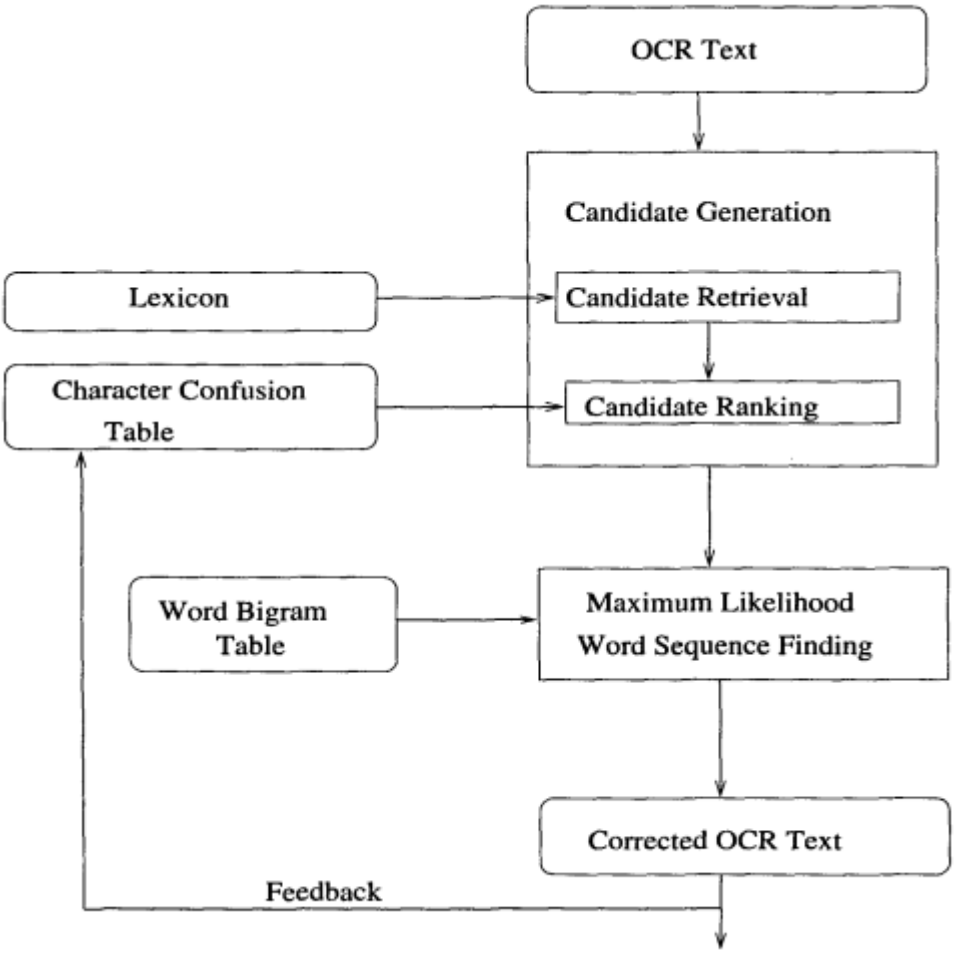

Рисунок 1.14 – Архітектура системи корекції слів для постобробки OCR

Словник генерується з навчального тексту; він включає всі слова з навчального набору, частота яких перевищує заданий поріг. Слова у словнику індексуються літерними n-грамами, як описано у попередньому розділі. Загальний процес виправлення речення виглядає наступним чином:

1. Прочитайте речення з вхідного тексту розпізнавання.

2. Для кожної можливої помилки витягнути зі словника до M кандидатів J. Переранжувати M кандидатів за їхніми умовними ймовірностями помилки.

Залишити тільки N найкращих кандидатів для наступного кроку обробки. (У поточній системі M дорівнює 10 000, а N – 10.)

3. Використати алгоритм Вітербі, щоб отримати найкращу послідовність слів для рядків у реченні.

Рис. 1.15 ілюструє альтернативні варіанти та оптимальний шлях, знайдений під час обробки (виправлення) речення «john fornd he man».

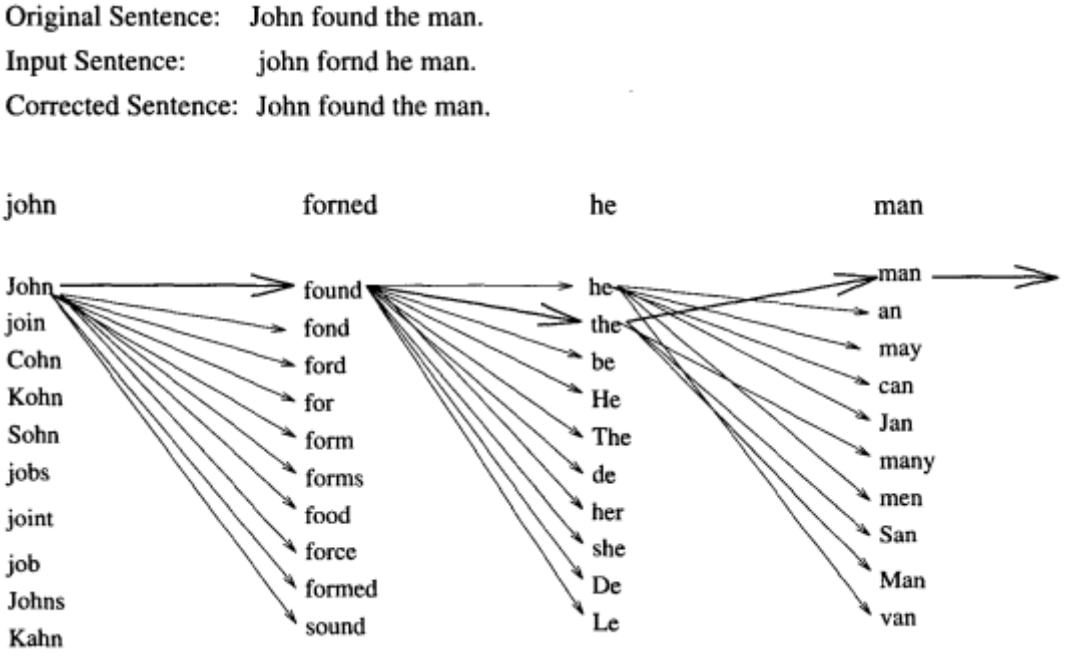

Рисунок 1.15 – Пошук оптимального вихідного результату

Для виправлення OCR-тексту системі потрібно кілька проходів. На першому проході система не має інформації про ймовірність переплутування символів, тому вона прийме попереднє переконання o~ як ймовірність того, що символ буде правильно розпізнано. Решту ймовірності система розподіляє рівномірно між іншими подіями.

На кожному кроці зворотного зв'язку система спочатку генерує таблицю ймовірності переплутування символів, порівнюючи текст OCR з виправленим текстом OCR з останнього проходу. Вона використовує нову таблицю плутанини для виправлення тексту під час наступного проходу [20].

1.7. Висновки до Розділу І

Висновки до теоретичного розділу дипломної роботи підсумовують основні аспекти, які було розглянуто в дослідженні, а саме:

Огляд існуючої літератури про розпізнавання тексту із зображень дозволив встановити, що OCR-технології є основоположним інструментом для автоматизації обробки текстових даних з різних джерел. Було виявлено, що сучасні підходи до OCR суттєво еволюціонували від базових методів до складних алгоритмів машинного навчання та глибоких нейронних мереж, що забезпечують високу точність і ефективність розпізнавання.

Основні поняття розпізнавання тексту із зображень (OCR) розкрили фундаментальні принципи, на яких базується процес обробки зображень для вилучення текстових даних. Було розглянуто ключові етапи роботи систем OCR, такі як попередня обробка зображення, сегментація, розпізнавання символів, а також постобробка для покращення точності результату.

Класифікація систем розпізнавання символів показала різноманітність підходів до OCR, включаючи традиційні методи, які використовують шаблони для розпізнавання символів, та сучасні системи, що базуються на машинному навчанні. Встановлено, що найвищу точність забезпечують системи, які застосовують гібридні підходи, комбінуючи різні методи для розпізнавання та корекції тексту.

Необхідний набір даних для розпізнавання тексту та перевірка ASCII-відповідності зображень є важливими аспектами для успішної реалізації OCR-систем. Аналіз показав, що якість вхідних даних значною мірою впливає на кінцеву точність розпізнавання. Для ефективної роботи систем необхідно забезпечити наявність чистих, добре структурованих зображень, що відповідають стандартам кодування символів, зокрема ASCII.

Алгоритми розпізнавання тексту на основі машинного навчання є ключовими для сучасних OCR-систем. Було розглянуто кілька підходів, таких як згорткові нейронні мережі (CNN) і рекурентні нейронні мережі (RNN), які

значно підвищують точність розпізнавання тексту, особливо при обробці складних або нерегулярних текстових зображень. Використання цих алгоритмів дозволяє адаптувати системи до різних типів шрифтів, мов та стилів.

Корекція слів для постобробки OCR є важливим етапом для забезпечення високої якості результатів. Після розпізнавання символів алгоритми корекції допомагають виправляти помилки, що виникають через схожість символів або дефекти зображень. Це дозволяє значно підвищити точність тексту, особливо у випадках роботи з шумними або пошкодженими зображеннями.

Таким чином, теоретичний аналіз основних аспектів OCR показав, що розвиток програмного забезпечення для розпізнавання тексту із зображень вимагає комплексного підходу. Використання передових алгоритмів машинного навчання, забезпечення якісного набору даних та корекція помилок після обробки дозволяють створювати високоточні та надійні OCR-системи, які можуть успішно застосовуватися у різних галузях, таких як архівування документів, автоматизація бізнес-процесів і обробка великих обсягів текстової інформації.

РОЗДІЛ ІІ
ПРОЕКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕПЕЗПЕЧЕННЯ

2.1 Функціональні вимоги до ПЗ

Функціональні вимоги – це особливості продукту або функції, які розробники повинні реалізувати, щоб користувачі могли виконувати свої завдання. Тому важливо зробити їх зрозумілими як для команди розробників, так і для зацікавлених сторін. Зазвичай функціональні вимоги описують поведінку системи за певних умов.

Функціональні вимоги будуть відрізнятися для різних типів програмного забезпечення. Наприклад, функціональні вимоги для веб-сайту або мобільного додатку повинні визначати потоки користувачів і різні сценарії взаємодії. Прикладами вимог можуть бути такі:

- система надсилає електронного листа з підтвердженням, коли створюється новий обліковий запис користувача;
- система надсилає запит на затвердження після того, як користувач вводить особисту інформацію;
- функція пошуку дозволяє користувачам шукати вміст/товари, вводячи запит у пошуковий рядок;
- користувач може переглянути товари в кошику, змінити їх кількість або видалити перед оформленням замовлення [21].

Для розроблюваного програмного забезпечення для розпізнавання тексту із зображень виділено такі функціональні вимоги, які необхідні для коректної роботи:

Ось деякі функціональні вимоги для програмного забезпечення, яке реалізує зазначений вами код для розпізнавання тексту з зображень:

1. Завантаження зображення:

- підтримка завантаження зображень у різних форматах (JPEG, PNG, BMP тощо);
- відображення помилки у випадку невдалого завантаження зображення (наприклад, якщо файл не знайдено).

2. Попередня обробка зображення:

Перетворення завантаженого зображення на відтінки сірого для покращення якості розпізнавання.

- застосування Gaussian Blur для зменшення шуму на зображенні;
- використання CLAHE (Contrast Limited Adaptive Histogram Equalization) для підвищення контрасту зображення;
- бінаризація зображення (перетворення в чорно-біле) з метою підготовки його для розпізнавання тексту.

3. Розпізнавання тексту:

- використання Tesseract OCR для розпізнавання тексту на бінарному зображенні;
- підтримка розпізнавання тексту англійською мовою;
- виведення розпізнаного тексту в консоль з форматуванням для кращої читабельності (з абзацами або рядками).

4. Візуалізація результатів:

- відображення оригінального зображення разом з обробленим зображенням у графічному інтерфейсі;
- відображення контурів текстових блоків на оригінальному зображенні;
- відображення розпізнаного тексту над відповідними текстовими блоками на зображенні.

5. Збереження результатів:

- збереження зображення з виділеними блоками тексту та з розпізнаним текстом у файли.

6. Обробка помилок:

- виведення відповідних повідомлень про помилки користувачу.

2.2 Обґрунтування вибору мови програмування для розробки ПЗ

Для розробки програмного забезпечення для розпізнавання тексту із зображень було обрано мову програмування Python (версії 3.12.0)

Python – це високорівнева інтерпретована мова програмування з простим синтаксисом, що робить її легко читабельною та надзвичайно зручною для користувачів і початківців. Спочатку створена для задоволення бажання Гвідо Ван Россума створити просту у використанні та красиву на вигляд мову програмування, Python була вперше випущена у світ у 1991 році.

З часу свого розвитку вона набула широкого застосування серед розробників, аналітиків даних, дослідників тощо.

Оскільки Python є універсальною мовою, її можна використовувати в різноманітних додатках, а її нескладна природа робить її чудовою мовою для автоматизації завдань, створення веб-сайтів або програмного забезпечення та аналізу даних.

Python також має кілька інших характеристик, які роблять її популярною серед розробників та інженерів. До них відносяться:

- її легко читати. У коді Python використовуються англійські ключові слова, а не розділові знаки, а розриви рядків допомагають визначити блоки коду. На практиці це означає, що ви можете визначити, для чого призначений код, просто подивившись на нього;
- це відкритий код. Ви можете завантажити вихідний код, змінити його і використовувати як завгодно;
- його можна переносити. Деякі мови вимагають модифікації коду для роботи на різних платформах, але Python є кросплатформенною мовою, що означає, що ви можете запускати той самий код на будь-якій операційній системі за допомогою інтерпретатора Python;

- вона розширювана. Код на Python можна писати іншими мовами (наприклад, C++), а користувачі можуть додавати низькорівневі модулі до інтерпретатора Python для кастомізації та оптимізації своїх інструментів;
- він має широку стандартну бібліотеку. Ця бібліотека доступна будь-кому і означає, що користувачам не потрібно писати код для кожної окремої функції – вони можуть отримати доступ до вбудованих модулів, які допомагають вирішувати проблеми у повсякденному програмуванні тощо.

Python є популярною мовою для розробки веб-сайтів та програмного забезпечення, оскільки ви можете створювати складні, багатопротокольні додатки, зберігаючи при цьому стислий, читабельний синтаксис. Насправді, деякі з найпопулярніших додатків були створені за допомогою Python. Крім того, спільнота з відкритим вихідним кодом Python надає розробникам велику кількість коду, фреймворків та підтримки для багаторазового використання. Погляньмо на приклад: Django – один з найбільш використовуваних фреймворків Python, створений досвідченими розробниками, щоб допомогти іншим прискорити час створення додатків та уникнути проблем, які можуть завадити їхньому прогресу.

Однією з ключових переваг Python є його здатність автоматизувати ручні, повторювані завдання. За допомогою Python ви можете навчитися автоматизувати практично все, використовуючи вбудовані модулі або заздалегідь написаний код з його надійної бібліотеки. Або ж ви можете писати власні кастомні скрипти для виконання певних дій. Наприклад, ви можете легко автоматизувати надсилання електронних листів за допомогою модуля «smtplib» або копіювати файли за допомогою модуля «shutil».

Python також має потужний набір фреймворків для тестування, що робить її чудовою мовою для автоматизації тестування. Такі фреймворки, як Pytest, Behave та Robot дозволяють розробникам писати прості, але ефективні тести для забезпечення якості своїх збірок.

Подібно до того, як Python може допомогти аналітикам даних у роботі з великими масивами даних, Python широко використовується у фінансовій

індустрії для швидкого виконання складних обчислень. Фондові ринки генерують величезні обсяги даних, і Python можна використовувати для імпорту даних про ціни на акції та створення стратегій за допомогою алгоритмів для виявлення торгових можливостей. Мову також можна використовувати для оптимізації портфеля, управління ризиками, фінансового моделювання та візуалізації, аналізу криптовалют і навіть для виявлення шахрайства.

Python також можна знайти в деяких найскладніших технологіях штучного інтелекту (ШІ) – і це насправді одна з найкращих мов для ШІ. Лаконічний і читабельний код Python дозволяє розробникам створювати послідовні, надійні системи, а його велика бібліотека надає ряд фреймворків, таких як PyBrain, який пропонує розробникам потужні алгоритми для завдань машинного навчання.

Крім того, можливості візуалізації Python можуть допомогти перетворити ці великі набори даних для ШІ або МН у зрозумілі графіки або звіти. Цікаво, що OpenAI, дослідницька лабораторія штучного інтелекту, використовує фреймворк Python, Pytorch, як стандартний фреймворк для глибокого навчання, на якому тренує свої системи ШІ [22].

Підсумовуючи, мова програмування Python має багато переваг для проекту, пов'язаного з обробкою зображень і розпізнаванням тексту. По-перше, його простота та зрозумілість роблять код легким для читання і підтримки. По-друге, Python має потужні бібліотеки, такі як OpenCV і pytesseract, які значно спрощують реалізацію складних алгоритмів обробки зображень і OCR. Також Python забезпечує швидку розробку завдяки своїм динамічним можливостям і активному співтовариству, що сприяє легкому доступу до ресурсів і підтримки.

Це робить Python відмінним вибором для швидкої прототипізації і реалізації проектів у галузі комп'ютерного зору.

2.3 Обґрунтування вибору середовища для розробки ПЗ

Для розробки програмного забезпечення для розпізнавання тексту із зображень було обрано Google Colaboratory (Google Colab) як середовище розробки.

Google Colaboratory – це безкоштовне хмарне середовище для ноутбуків Jupyter, яке дозволяє нам тренувати наші моделі машинного навчання та глибокого навчання на CPU, GPU та TPU.

Не має значення, який у вас комп'ютер, яка його конфігурація і наскільки він старий. Ви все одно можете використовувати Google Colab Notebook! Все, що вам потрібно – це обліковий запис Google і веб-браузер. І ось вишенька на торті – ви отримуєте доступ до графічних процесорів, таких як Tesla K80 і навіть TPU, безкоштовно.

TPU набагато дорожчі за GPU, а на Colab ви можете користуватися ними безкоштовно. Варто повторювати знову і знову – це унікальна пропозиція.

У Google Colab блокнот – це веб-середовище для створення та запуску коду. Блокноти схожі на скрипти або файли коду в інших середовищах програмування, але мають деякі унікальні переваги. Блокноти дозволяють писати і виконувати код у веб-браузері, відображаючи результати в режимі реального часу. Це полегшує ітерації коду та візуалізацію результатів у процесі роботи. Блокноти Colab також підтримують розмітку, що дозволяє включати відформатований текст, рівняння та зображення поряд з кодом. Ви також можете додавати коментарі та нотатки до коду, що полегшує його розуміння та співпрацю з іншими. Загалом, блокноти є потужним інструментом для дослідників даних і фахівців з машинного навчання, забезпечуючи гнучке та інтерактивне середовище для написання і тестування коду.

Деякі з переваг використання Google Colab:

- доступність: Google Colab доступний з будь-якого веб-браузера, тому вам не потрібно встановлювати жодного програмного забезпечення на

свій комп'ютер. Ви можете отримати доступ до своїх блокнотів з будь-якої точки світу, якщо у вас є підключення до Інтернету. Ці особливості Google Colab роблять його зручним інструментом для кожного;

- потужність: Google Colab надає доступ до потужних обчислювальних ресурсів, включаючи GPU і TPU. Це означає, що ви можете навчати і запускати складні моделі машинного навчання швидко і ефективно;
- співпраця: Google Colab дозволяє легко співпрацювати з іншими над проектами. Ви можете ділитися своїми блокнотами з іншими, а вони можуть редагувати і запускати ваш код в режимі реального часу;
- освіта: блокнот Google Colab – чудовий інструмент для вивчення машинного навчання та науки про дані. Багато навчальних посібників і ресурсів доступні в Інтернеті, а також ви можете створювати і ділитися своїми блокнотами [23].

Само тому програмне забезпечення Google Colab ідеально підходить для застосунку, розроблюваного в даній роботі.

2.4 Архітектура програмного забезпечення

Архітектура програмного забезпечення системи являє собою проектні рішення, пов'язані із загальною структурою та поведінкою системи. Архітектура допомагає зацікавленим сторонам зрозуміти і проаналізувати, як система досягне таких важливих якостей, як модифікованість, доступність і безпека.

Архітектура програмного забезпечення підтримує аналіз якостей системи, коли команди приймають рішення щодо системи, а не після впровадження, інтеграції чи розгортання. Незалежно від того, чи розробляється нова система, чи розвивається успішна система, чи модернізується застаріла система, цей своєчасний аналіз дозволяє командам визначити, чи обрані ними підходи дадуть прийнятне рішення. Ефективна архітектура слугує концептуальним клеєм, який об'єднує всі зацікавлені

сторони на кожному етапі проекту, забезпечуючи гнучкість, економію часу та коштів, а також раннє виявлення проектних ризиків.

Побудова ефективної архітектури, яка дозволяє швидко створювати продукт для задоволення поточних потреб, а також досягати довгострокових цілей, може виявитися непростим завданням. Нездатність визначити, розставити пріоритети та знайти компроміси між архітектурно важливими якостями часто призводить до затримок проекту, дорогих доопрацювань або ще гірше.

Ефективна архітектура програмного забезпечення, що підтримується практиками гнучкої архітектури, забезпечує ефективну безперервну еволюцію системи. Такі практики включають документування архітектурних елементів і взаємозв'язків, призначених для досягнення ключових якостей; постійне оцінювання архітектури на відповідність бізнес-цілям і місії організації; аналіз розгорнутої системи на відповідність архітектурі. При правильному виконанні ці практики забезпечують передбачувану якість продукту, меншу кількість подальших проблем, економію часу та коштів на інтеграцію та тестування, а також економічно ефективну еволюцію системи [24].

Для розроблюваного в даній роботі програмного забезпечення можна виділити такі модулі загальної архітектури:

Архітектура програми для розпізнавання тексту на зображенні за допомогою Python і бібліотек OpenCV та Tesseract може бути поділена на кілька основних модулів, кожен з яких виконує певний етап обробки зображення та розпізнавання тексту. Цей підхід забезпечує гнучкість і масштабованість при розробці подібних рішень.

1. Модуль завантаження зображення. Програма починається з завантаження вхідного зображення за допомогою бібліотеки OpenCV. Важливою частиною цього модуля є перевірка на успішність завантаження файлу, щоб уникнути можливих помилок, таких як некоректний шлях до файлу або відсутність зображення. Якщо зображення не завантажується,

користувач отримує відповідне повідомлення про помилку. Таким чином, програма одразу зупиняється на ранньому етапі, якщо вхідні дані недоступні.

2. Модуль попередньої обробки зображення. Після завантаження зображення здійснюється його попередня обробка. Це включає кілька послідовних етапів:

- перетворення в градації сірого: Вхідне зображення перетворюється з кольорового формату BGR в сірий (градації сірого), що значно спрощує подальшу обробку, оскільки позбавляє зайвих кольорових інформацій;
- зменшення шуму: Використання методу Gaussian Blur для зменшення шумів та артефактів, які можуть ускладнити процес розпізнавання тексту;
- підвищення контрасту: Застосування адаптивної гістограми (CLAHE), яка покращує контраст зображення, роблячи текст більш виразним на фоні зображення.
- бінаризація: Перетворення зображення у двоколірний формат (чорно-біле) за допомогою методу порогової обробки, що підходить для більш точного розпізнавання тексту.

3. Модуль візуалізації зображень. Для наочності програма виводить два зображення: оригінальне та оброблене (бінаризоване). Це дозволяє користувачу порівняти результати попередньої обробки та візуально оцінити, як вплинули етапи перетворення на якість зображення. Для цього використовуються subplot'и з бібліотеки Matplotlib, що дозволяють вивести кілька зображень поруч.

4. Модуль розпізнавання тексту. Основний функціонал програми базується на бібліотеці Tesseract, яка здійснює розпізнавання тексту з обробленого зображення. Програма отримує структуру даних із інформацією про кожен блок тексту (координати, розміри, сам текст) у форматі словника. Передбачено обробку можливих помилок на цьому етапі: якщо Tesseract не

вдається розпізнати текст через певні технічні причини, програма виводить повідомлення про помилку і припиняє виконання.

5. Модуль виведення розпізнаного тексту. Після успішного розпізнавання програма виводить текст, розділяючи його на рядки для кращої читабельності. У разі, якщо текст не розпізнаний, виводиться відповідне повідомлення. Це забезпечує користувача результатом навіть у випадках, коли зображення містить низькоякісний або відсутній текст.

6. Модуль відображення контурів блоків тексту. Програма також дає можливість візуально побачити блоки тексту на оригінальному зображенні. Для кожного блоку тексту малюється прямокутник, який показує, де саме був розпізнаний текст. Це виконується у двох варіантах:

- прямокутники навколо текстових блоків: Виділення кожного текстового блоку, що дозволяє побачити, де Tesseract знайшов текст;
- текст над блоками: Виділення блоків тексту разом з відображенням самого тексту над ними, що є корисним для візуальної перевірки якості розпізнавання.

7. Модуль збереження результатів. Програма зберігає результати у вигляді двох окремих зображень:

- зображення з прямокутниками навколо текстових блоків;
- зображення з текстом, який накладено над відповідними блоками.

8. Інтерактивний інтерфейс. Програма має інтерфейс командного рядка, де користувач отримує повідомлення про можливі помилки або інструкції щодо успішного завершення кожного етапу обробки. Виведення тексту відбувається безпосередньо в консолі, що дозволяє швидко переглянути результат.

Ця архітектура поєднує автоматизацію обробки зображень, текстового аналізу і візуалізації результатів, забезпечуючи чітку та ефективну роботу для проектів із розпізнавання тексту.

2.5 Програмна реалізація застосунку

Програмне забезпечення, розроблюване в даній дипломній роботі, виконує розпізнавання тексту на зображенні за допомогою бібліотеки «pytesseract» та бібліотек для комп'ютерного зору й обробки зображень. Основні блоки коду включають:

1. Інсталяція та імпорт бібліотек (див. Рис. 2.5.1-2.5.2):

«!pip install pytesseract», «!sudo apt install tesseract-ocr» – інсталяція необхідних компонентів для роботи з Tesseract OCR.

«import cv2», «import pytesseract», «import matplotlib.pyplot as plt», «from pylab import rcParams», «from pytesseract import Output» – імпорт бібліотек для обробки зображень («cv2»), оптичного розпізнавання символів («pytesseract»), візуалізації результатів («matplotlib»), та форматування виводу («Output»).

```
!pip install pytesseract
!sudo apt install tesseract-ocr
```

Рисунок 2.1 – Інсталяція та імпорт бібліотек

```
import cv2
import pytesseract
import matplotlib.pyplot as plt
from pylab import rcParams
from pytesseract import Output
```

Рисунок 2.2 – Інсталяція та імпорт бібліотек

2. Налаштування параметрів візуалізації (див. Рис. 2.3):

«rcParams['figure.figsize'] = 16, 8» – задає розмір фігур (графіків) для відображення зображень у фіксованому розмірі.

```
# Параметр бібліотеки matplotlib, який задає розміри фігури (графіка)
rcParams['figure.figsize'] = 16, 8
```

Рисунок 2.3 – Налаштування параметрів візуалізації

3. Завантаження зображення (див. Рис. 2.4):

«file_name = «image1.jpg» – визначення шляху до зображення.

«img = «cv2.imread(file_name)» – завантаження зображення з використанням бібліотеки OpenCV.

Перевірка на успішне завантаження зображення. Якщо зображення не знайдено, виводиться повідомлення про помилку.

```
# Завантаження зображення
file_name = "image1.jpg"
img = cv2.imread(file_name)
```

Рисунок 2.4 – Завантаження зображення

4. Попередня обробка зображення (див. Рис. 2.5):

Перетворення зображення на відтінки сірого: «cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)». Це полегшує подальшу обробку, оскільки алгоритми розпізнавання тексту краще працюють із сірими або чорно-білими зображеннями.

Зменшення шуму: «cv2.GaussianBlur(gray, (5, 5), 0)» – згладжує зображення за допомогою гаусового фільтра для зменшення шумів, що покращує точність розпізнавання.

Підвищення контрасту: «clahe = cv2.createCLAHE()» і «contrast = clahe.apply(blur)» – алгоритм адаптивного рівняння гістограми покращує контрастність на різних ділянках зображення.

Бінаризація: «_, binary_img = cv2.threshold()» – перетворення зображення у чорно-білий формат для полегшення розпізнавання тексту.

```
# Перевірка на успішне завантаження зображення
if img is None:
    print(f"Помилка: Не вдалося завантажити зображення '{file_name}'. Перевірте шлях до файлу.")
else:
    # Попередня обробка зображення

    # Перетворення на відтінки сірого
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Зменшення шуму за допомогою Gaussian Blur
    blur = cv2.GaussianBlur(gray, (5, 5), 0)

    # Підвищення контрасту за допомогою CLAHE
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
    contrast = clahe.apply(blur)

    # Бінаризація зображення (чорно-біле перетворення)
    _, binary_img = cv2.threshold(contrast, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
```

Рисунок 2.5 – Попередня обробка зображення

5. Візуалізація зображень (див. Рис. 2.6):

Використовується «matplotlib» для відображення оригінального та обробленого зображень. Оригінальне зображення показується поруч з бінаризованим (обробленим) зображенням, що полегшує порівняння.

```
# Виведення оригінального та обробленого зображень з підписами
fig, axs = plt.subplots(1, 2, figsize=(16, 8))

# Оригінальне зображення
axs[0].imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
axs[0].set_title('Оригінальне зображення')
axs[0].axis('off')

# Оброблене зображення (чорно-біле)
axs[1].imshow(binary_img, cmap='gray')
axs[1].set_title('Оброблене чорно-біле зображення')
axs[1].axis('off')

plt.show()
```

Рисунок 2.6 – Візуалізація зображень

6. Розпізнавання тексту (див. Рис. 2.7):

«pytesseract.image_to_data(binary_img, output_type=Output.DICT)» – витягує дані про текстові блоки (координати блоків, текст, розмір та інше) з

бінаризованого зображення у форматі словника.

Перевірка та виведення тексту: Весь розпізнаний текст об'єднується у строку й виводиться порядково, з перевіркою на порожні рядки.

```
# Отримання інформації про текстові блоки
try:
    data = pytesseract.image_to_data(binary_img, output_type=Output.DICT)
except pytesseract.TesseractError as e:
    print(f"Помилка при розпізнаванні тексту: {e}")
    exit(1)

# Виведення розпізнаного тексту
recognized_text = " ".join(data['text']).strip()
if recognized_text:
    print("Розпізнаний текст:")
    # Розділення тексту на рядки
    for line in recognized_text.splitlines():
        if line.strip():  # Перевірка, чи рядок не пустий
            print(line)  # Вивід кожного рядка
else:
    print("Текст не був розпізнаний.")

# Відображення контурів блоків тексту на оригінальному зображенні (для лівого subplot)
n_boxes = len(data['text'])
img_with_boxes = img.copy()
for i in range(n_boxes):
    if data['text'][i].strip() != "":  # Перевірка, чи блок містить текст
        (x, y, w, h) = (data['left'][i], data['top'][i], data['width'][i], data['height'][i])
        # Малювання прямокутників навколо тексту
        img_with_boxes = cv2.rectangle(img_with_boxes, (x, y), (x + w, y + h), (0, 255, 0), 2)

# Відображення контурів блоків та тексту над кожним блоком (для правого subplot)
img_with_text = img.copy()
for i in range(n_boxes):
    if data['text'][i].strip() != "":  # Перевірка, чи блок містить текст
        (x, y, w, h) = (data['left'][i], data['top'][i], data['width'][i], data['height'][i])
        # Малювання прямокутників навколо тексту
        img_with_text = cv2.rectangle(img_with_text, (x, y), (x + w, y + h), (0, 255, 0), 2)
        # Додавання розпізнаного тексту над прямокутником
        img_with_text = cv2.putText(img_with_text, data['text'][i], (x, y - 10),
                                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 0, 0), 2)
```

Рисунок 2.7 – Розпізнавання тексту

7. Візуалізація блоків з текстом (див. Рис. 2.8):

Малювання прямокутників навколо текстових блоків: За координатами кожного текстового блоку малюються прямокутники на оригінальному зображенні («cv2.rectangle»).

Додавання тексту: Поруч із кожним прямокутником малюється текст, що був розпізнаний, з використанням «cv2.putText».

```
# Відображення двох subplot: лівий – тільки з прямокутниками, правий – з текстом
fig, axs = plt.subplots(1, 2, figsize=(16, 8))

# Ліве зображення – з прямокутниками навколо блоків
axs[0].imshow(cv2.cvtColor(img_with_boxes, cv2.COLOR_BGR2RGB))
axs[0].set_title("Зображення з виділеними блоками")
axs[0].axis('off')

# Праве зображення – з прямокутниками та текстом над ними
axs[1].imshow(cv2.cvtColor(img_with_text, cv2.COLOR_BGR2RGB))
axs[1].set_title("Зображення з текстом над блоками")
axs[1].axis('off')

plt.show()
```

Рисунок 2.8 – Візуалізація блоків з текстом

8. Виведення результатів (див. Рис. 2.9):

Візуалізація двох зображень: з прямокутниками навколо текстових блоків та з текстом над блоками.

Збереження результатів: Збереження зображень із текстовими блоками та підписами у файли «image_with_boxes.jpg» та «image_with_text.jpg».

```
# Збереження результатів
cv2.imwrite("image_with_boxes.jpg", img_with_boxes)
cv2.imwrite("image_with_text.jpg", img_with_text)
print("Результати збережено у файлах 'image_with_boxes.jpg' та 'image_with_text.jpg'.")
```

Рисунок 2.9 – Виведення результатів

Повний код розробленого застосунку наведено в Додатку А.

2.6 Функціонування програмного забезпечення

Функціонування розроблюваного програмного забезпечення спрямоване на автоматизацію процесу розпізнавання тексту з зображень з використанням сучасних методів обробки зображень та технологій оптичного розпізнавання символів. Це програмне забезпечення дозволяє користувачеві завантажувати зображення, автоматично обробляти його для поліпшення якості та

здійснювати точне виділення текстових блоків. Основним завданням є перетворення зображень у текстовий формат з метою подальшої обробки чи збереження даних.

Безпосередня робота програмного забезпечення починається із завантаження файлу-зображення, текст на якому треба розпізнати. Цей процес у середовищі Google Colab продемонстровано на Рис. 2.10-2.12.

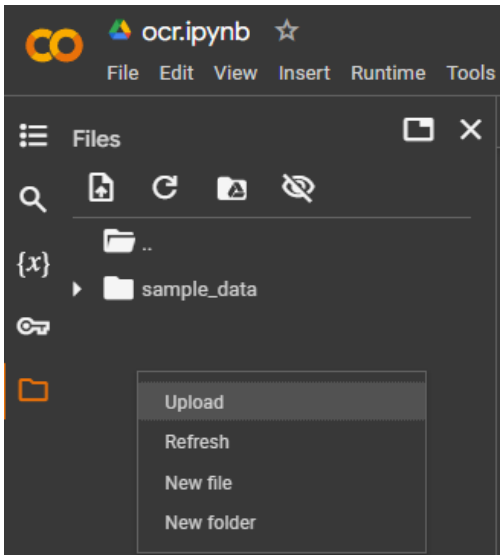

Рисунок 2.10 – Робота із файлами в Google Colab

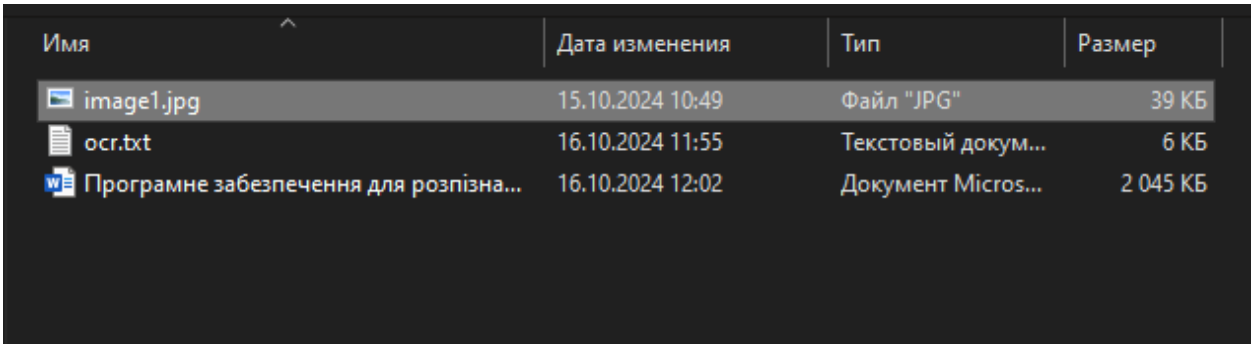

Рисунок 2.11– Робота із файлами в Google Colab

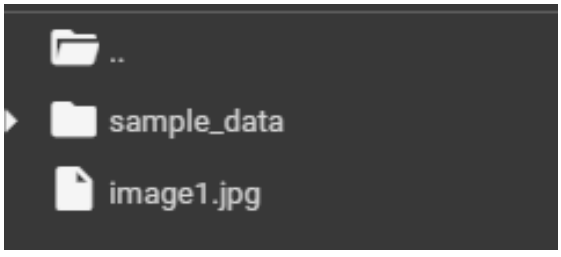

Рисунок 2.12 – Робота із файлами в Google Colab

Після завантаження файлу, можна запустити програму, натиснувши на відповідну кнопку, як показано на Рис. 2.13.

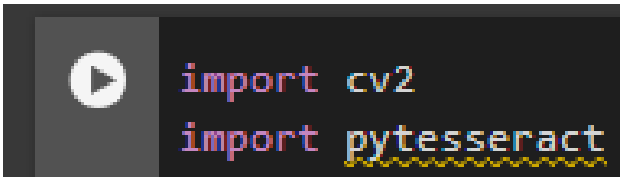

Рисунок 2.13 – Кнопка запуску програми

Після запуску розробленого програмного забезпечення, у консоль виводяться результати роботи застосунку. Вони включають в себе вивід sublot’a, що містить оригінальне зображенні і зображення з передобробкою, розпізнаний текст, sublot’a що містить оригінальне зображення із виділеними блоками тексту і зображення із виділеними блоками тексту і доданим розпізнаним текстом, як показано на Рис. 2.14-2.16.

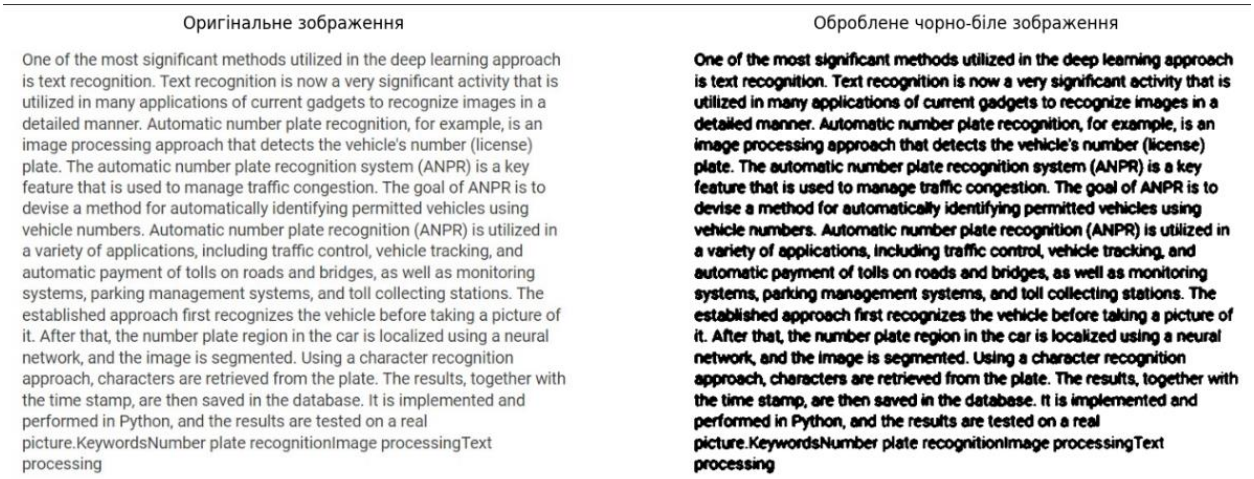

Рисунок 2.14 – Вивід програмою sublot’a, що містить оригінальне зображенні і зображення з передобробкою

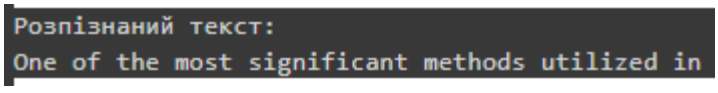

Рисунок 2.15 – Вивід програмою розпізнаного тексту

Рисунок 2.16 – Вивід програмою sublot'а що містить оригінальне зображення із виділеними блоками тексту і зображення із виділеними блоками тексту і доданим розпізнаним текстом

Крім того, програмне забезпечення створює два файли-зображення з виділеними блоками і текстом, текстовий файл із розпізнаним текстом, як показано на Рис. 2.17-2.20.

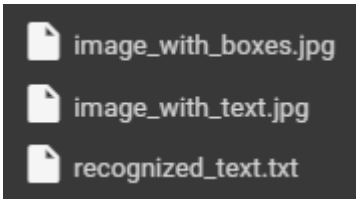

Рисунок 2.17 – Створені файли

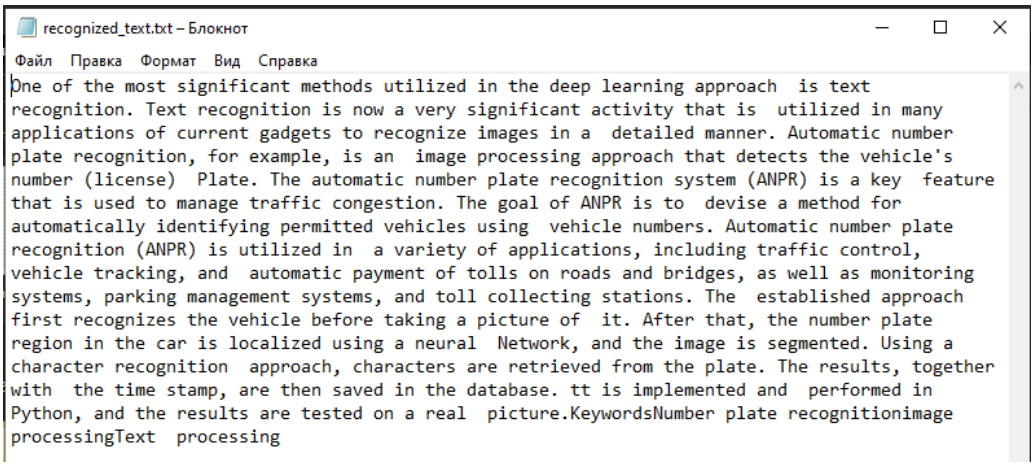

Рисунок 2.18 – Розпізнаний текст

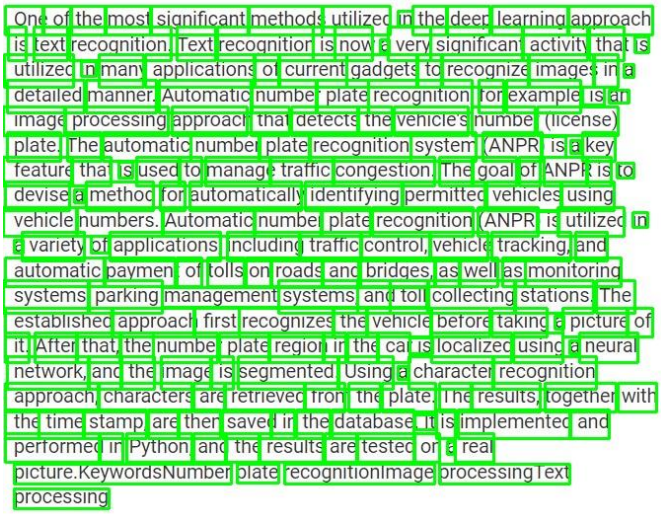

Рисунок 2.19 – Зображення із виділеними розпізнаними текстовими блоками

Рисунок 2.20 – Зображення із виділеними розпізнаними текстовими блоками і доданим текстом

2.7 Висновки до Розділу II

У цьому розділі було детально розглянуто основні етапи проектування та програмної реалізації програмного забезпечення для розпізнавання тексту з зображень. Визначення функціональних вимог до програмного забезпечення заклало основу для подальшого проектування, зокрема, дозволило зосередитися на найважливіших аспектах, таких як обробка зображень, розпізнавання тексту та збереження результатів.

Обґрунтування вибору мови програмування підтвердило, що Python є оптимальним варіантом завдяки своїм широким можливостям роботи з бібліотеками для обробки зображень та розпізнавання тексту. Аналіз різних середовищ розробки показав, що обране середовище забезпечує комфортну роботу, зручні інструменти для налагодження та тестування, що значно спростило процес розробки.

Опис архітектури програмного забезпечення продемонстрував логічну структуру, яка дозволяє ефективно організувати роботу програми та забезпечити легкість подальшого вдосконалення і підтримки. Реалізація основних функцій програми відповідала визначеним вимогам, що підтверджується успішним проходженням тестування, яке включало ключові аспекти роботи програми.

Висвітлення процесу функціонування програмного забезпечення продемонструвало його здатність обробляти зображення, розпізнавати текст та надавати результати у зручному для користувача форматі.

РОЗДІЛ ІІІ
ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Сценарії тестування програмного забезпечення

Тестовий сценарій – це документ, який дуже коротко, переважно в одному рядку, описує всю функціональність програмного забезпечення з точки зору кінцевого користувача.

Сценарій призначений для моделювання реальної ситуації, з якою користувач зіткнеться під час використання програмного забезпечення. Цей документ фокусується на тому, що потрібно перевірити, допомагаючи команді тестувальників контролювати процес тестування. Завдяки тому, що тестові сценарії базуються на високому рівні дій і містять короткий опис, їх досить просто створювати і оновлювати в режимі реального часу.

Тестові сценарії також допомагають економити час і підходять компаніям і командам, які працюють за гнучкими методологіями.

Кожен член команди тестування може легко відкрити документ і зрозуміти, які тести виконані, і почати працювати над наступним сценарієм. Цей документ дозволяє тестувальникам сконцентруватися на процедурі тестування, замість того, щоб записувати багато даних у складний файл.

Наприклад, правильним тестовим сценарієм може бути «Перевірка функціональності входу в систему програмного забезпечення» [25].

Для розробленого програмного забезпечення виділено такі основні тестові сценарії у вигляді Таблиць 3.1-3.9.

Таблиця 3.1 – Тестові сценарії для розробленого застосунку

Елемент тестування	Опис
Мета	Перевірити, чи може програма коректно завантажити зображення.
Вхідні дані	Коректний шлях до файлу зображення.
	Некоректний шлях до файлу.
Очікуваний результат	Коректний шлях: зображення успішно завантажується.
	Некоректний шлях: програма виводить повідомлення про помилку.
Фактичний результат	Коректний шлях: зображення завантажується успішно.
	Некоректний шлях: виведено повідомлення про помилку.

Таблиця 3.2 – Тестові сценарії для розробленого застосунку

Елемент тестування	Опис
Мета	Перевірити правильність обробки зображення (перетворення в сірий масштаб, фільтрація, бінаризація).
Вхідні дані	Зображення з високим і низьким контрастом.
Очікуваний результат	Оброблене зображення має бути чіткішим для зображень із низьким контрастом.
Фактичний результат	Контрастність зображень покращено, шум зменшено. Чіткість зображень з низьким контрастом підвищилась.

Таблиця 3.3 – Тестові сценарії для розробленого застосунку

Елемент тестування	Опис
Мета	Перевірити точність розпізнавання тексту з різних типів зображень.
Вхідні дані	Зображення з чітким друкованим текстом.
	Зображення з розмитим або рукописним текстом.
Очікуваний результат	Чіткий текст: програма розпізнає текст повністю.
	Низька якість тексту: текст частково розпізнаний або не розпізнаний.
Фактичний результат	Текст на зображеннях з чітким текстом повністю розпізнаний.
	Розмитий текст розпізнаний частково.

Таблиця 3.4 – Тестові сценарії для розробленого застосунку

Елемент тестування	Опис
Мета	Перевірити, чи зберігається розпізнаний текст у файл.
Вхідні дані	Зображення з текстом.
Очікуваний результат	Створюється файл recognized_text.txt, що містить розпізнаний текст.
Фактичний результат	Створено файл recognized_text.txt, текст збережено успішно.

Таблиця 3.5 – Тестові сценарії для розробленого застосунку

Елемент тестування	Опис
Мета	Перевірити, чи коректно програма виділяє текстові блоки на зображенні та зберігає результат.
Вхідні дані	Зображення з текстовими блоками.
Очікуваний результат	Створюються файли image_with_boxes.jpg і image_with_text.jpg з коректним виділенням блоків.
Фактичний результат	Створено файли image_with_boxes.jpg і image_with_text.jpg з правильним виділенням блоків.

Таблиця 3.6 – Тестові сценарії для розробленого застосунку

Елемент тестування	Опис
Мета	Перевірити, чи коректно виводяться повідомлення про помилки під час розпізнавання тексту.
Вхідні дані	Зображення, яке не містить тексту або містить невідомі символи.
Очікуваний результат	Програма виводить повідомлення, що текст не був розпізнаний.
Фактичний результат	Виведено повідомлення про відсутність розпізнаного тексту.

Таблиця 3.7 – Тестові сценарії для розробленого застосунку

Елемент тестування	Опис
Мета	Перевірити, як програма обробляє великі зображення з великою кількістю тексту.
Вхідні дані	Велике зображення з текстом на декількох сторінках.
Очікуваний результат	Програма успішно розпізнає текст без помилок та зберігає результат у файл.
Фактичний результат	Текст успішно розпізнаний та збережений.

Таблиця 3.8 – Тестові сценарії для розробленого застосунку

Елемент тестування	Опис
Мета	Перевірити, чи коректно відображаються блоки тексту після розпізнавання.
Вхідні дані	Зображення з текстом у різних областях.
Очікуваний результат	На зображенні чітко видно всі блоки з текстом, обведені прямокутниками.
Фактичний результат	Блоки з текстом чітко виділені на зображенні.

Таблиця 3.9 – Тестові сценарії для розробленого застосунку

Елемент тестування	Опис
Мета	Перевірити, як програма обробляє кольорові зображення з текстом.
Вхідні дані	Кольорове зображення з текстом різних шрифтів і кольорів.
Очікуваний результат	Програма успішно конвертує кольорове зображення у чорно-біле і розпізнає текст.
Фактичний результат	Зображення успішно оброблене, текст розпізнаний.

Ці сценарії тестування були успішно пройдені. Програма продемонструвала стабільну роботу за різних умов, таких як обробка зображень із текстом різної якості, розмірів, та контрасту. Система коректно завантажувала зображення, обробляла їх та виконувала розпізнавання тексту. Крім того, збереження результатів відбулося без помилок. Повідомлення про помилки також були виведені належним чином у передбачених випадках.

3.2. Чек лист для тестування програмного забезпечення

Контрольний список (чек лист) слугує практичним інструментом тестування, який організовує, впорядковує та прискорює процес контролю якості. Він складається з вичерпного каталогу або списку завдань, які необхідно виконати, забезпечуючи структурований підхід для відстеження прогресу. Коли кожен пункт у списку виконано, його відмічають, видаляють або ставлять галочку, щоб показати, що він виконаний.

Мета контрольного списку тестування програмного забезпечення – забезпечити всебічне охоплення під час тестування програмного забезпечення, чітко і систематично, використовуючи завдання або пункти. Контрольні списки зосереджені на ключових областях, функціях і діях, які необхідно виконати, і не містять покрокових інструкцій, як це зробити.

Контрольний список тестування програмного забезпечення зазвичай охоплює різні аспекти тестування, включаючи планування тестів, створення тестових кейсів, виконання тестів, управління дефектами та звітування. Дотримуючись контрольного списку, тестувальники можуть виявити потенційні проблеми на ранніх стадіях, дотримуватися стандартів тестування та найкращих практик і, в кінцевому підсумку, підвищити якість програмного забезпечення, яке тестується [26].

Чек лист необхідних перевірок для розробленого програмного забезпечення для розпізнавання тексту із зображень подано у вигляді Таблиці 3.10

Таблиця 3.10 – Чек лист для розробленого застосунку

№	Опис тесту	Очікуваний результат	Результат
1	Перевірка успішного завантаження зображення	Зображення успішно завантажується, помилки не виникають	Пройдено
2	Обробка кольорового зображення	Зображення перетворюється на чорно-біле для покращення розпізнавання	Пройдено
3	Застосування Gaussian Blur для зменшення шуму	Шум на зображенні зменшений, зображення стає чіткішим	Пройдено
4	Підвищення контрасту за допомогою CLAHE	Контраст зображення підвищується, текст стає більш видимим	Пройдено
5	Бінаризація зображення (перетворення на чорно-біле)	Зображення успішно перетворюється на чорно-біле	Пройдено
6	Розпізнавання тексту з обробленого зображення	Текст розпізнаний та виведений у консоль	Пройдено
7	Розпізнавання текстових блоків і відображення контурів	Навколо текстових блоків малюються прямокутники	Пройдено
8	Збереження зображення з контурами блоків	Зображення з прямокутниками навколо блоків успішно зберігається	Пройдено
9	Збереження розпізнаного тексту у текстовий файл	Файл .txt створений, розпізнаний текст записаний у файл	Пройдено
10	Виведення повідомлення про помилку, якщо текст не розпізнано	Якщо текст не розпізнано, виводиться відповідне повідомлення	Пройдено

Усі пункти тестування, зазначені у чек-листі, були успішно виконані. Програмне забезпечення для розпізнавання тексту зображень продемонструвало стабільну роботу в усіх ключових аспектах, включно з попередньою обробкою зображення, розпізнаванням тексту, відображенням текстових блоків та збереженням результатів у відповідні файли.

3.3. Висновки до Розділу III

У цьому розділі були проаналізовані основні етапи тестування програмного забезпечення, що дозволили оцінити його працездатність та ефективність. У підрозділі, присвяченому сценаріям тестування, було розроблено п'ять основних сценаріїв, які охоплюють ключові функції програми. Ці сценарії забезпечили систематичний підхід до перевірки різних аспектів роботи програмного забезпечення, включаючи завантаження зображень, обробку зображень, розпізнавання тексту та збереження результатів.

Чек-лист для тестування програмного забезпечення став важливим інструментом для забезпечення всебічного оцінювання функціональних можливостей програми. Він дозволив структуровано перевірити виконання всіх вимог, визначених на етапі проектування. Успішне проходження тестування підтвердило, що програма відповідає встановленим вимогам і функціонує стабільно.

Завдяки проведеному тестуванню було виявлено, що програмне забезпечення не тільки виконує всі визначені функції, але й здатне адекватно реагувати на різні умови, що можуть виникати під час роботи.

ВИСНОВКИ

У результаті виконання дипломної роботи було успішно розроблено програмне забезпечення для розпізнавання тексту на зображеннях, яке базується на методах обробки зображень та технологіях оптичного розпізнавання символів (OCR). У процесі дослідження були визначені основні функціональні вимоги, що дозволили створити ефективний інтерфейс та реалізувати ключові функції, такі як завантаження, обробка зображень і збереження розпізнаного тексту.

Обрані мова програмування та середовище розробки забезпечили високу продуктивність та надійність програмного продукту. Архітектура програмного забезпечення була спроектована з урахуванням модульності та масштабованості, що дозволило в подальшому вдосконалювати його функціональні можливості.

Важливим етапом роботи стало тестування програмного забезпечення, що підтвердило його стабільність та відповідність встановленим вимогам. Розроблені сценарії тестування та чек-лист забезпечили всебічну перевірку функціональності програми, що свідчить про її готовність до впровадження в практичну діяльність.

Виконана робота не лише сприяла досягненню поставлених цілей, але й відкрила нові перспективи для подальших досліджень у галузі обробки зображень та оптичного розпізнавання тексту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Handwritten Optical Character Recognition (OCR): A Comprehensive Systematic Literature Review (SLR) [Електронний ресурс] – Режим доступу до ресурсу: https://ieeexplore.ieee.org/abstract/document/9151144

2. Чаудхарі, А. «Optical Character Recognition Systems for English Language». Springer International Publishing AG, 2017.

3. Тіварі, С., Мішра, Ш., Бхатія, П., Ядав, П. К. «Optical Character Recognition using MATLAB». International Journal of Advanced Research in Electronics and Communication Engineering (IJARECE), 2013.

4. Манваткар, П. М., Ядав, С. Х. «Розпізнавання тексту на зображеннях». 2015 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS), Коїмбатор, 2015.

5. OPTICAL CHARACTER RECOGNITION [Електронний ресурс] – Режим доступу до ресурсу: http://ir.juit.ac.in:8080/jspui/bitstream/123456789/7102/1/Optical%20Character%20Recognition.pdf

6. Гонсалес, Р. С., Вудс, Р. Е. «Цифрова обробка зображень», 4-е видання, Pearson, 2018.

7. Н., Іслам, З., Нур, Н. «Огляд систем оптичного розпізнавання тексту», Journal of Information & Communication Technology-JICT, 2016.

8. Паль, А., Сінгх, Д. «Розпізнавання рукописних символів англійської мови за допомогою нейронної мережі», International Journal of Computer Science & Communication, 2010.

9. Дешмукх, С., Рага, Л. «Аналіз напрямкових ознак – штриху і контуру для розпізнавання рукописних символів», 2009.

10. Басу, Д., Мехер, С. «Розпізнавання рукописних символів», Джайпур, 2011.

11. What is Optical Character Recognition (OCR)? [Електронний ресурс] – Режим доступу до ресурсу: https://www.docsumo.com/blogs/ocr/definition

12. Хендерсон, В. Л., Маклін, Р. А. «Design of Experiments: A Realistic Approach». Marcel Dekker, Нью-Йорк, 1974.

13. Борсак, Дж., Хвей, Б. «Verification of GT1». Технічний звіт 94-05, Інститут інформаційних наук, Університет Невади, Лас-Вегас, Невада, 1994.

14. Каллан, Дж. П., Крофт, В. Б., Хардінг, С. М. «The INQUERY retrieval system», 2012.

15. Кейсі, Р. Г., Вонг, К. Й. «Image Analysis Applications». Marcel Dekker, Нью-Йорк.

16. Крофт, В. Б., Хардінг, С., Тагхва, К., Борсак, Дж. «An evaluation of information retrieval accuracy with simulated OCR output», 1994.

17. Evaluation of Model-Based Retrieval Effectiveness with OCR Text [Електронний ресурс] – Режим доступу до ресурсу: https://dl.acm.org/doi/pdf/10.1145/214174.214180

18. Review of Optical Character Recognition for Power System Image Based on Artificial Intelligence Algorithm [Електронний ресурс] – Режим доступу до ресурсу: https://cdn.techscience.cn/files/energy/2023/TSP_EE-120-3/TSP_EE_20342/TSP_EE_20342.pdf

19. Ван, Х., Ван, Х. «Improved approach based on SVM for license plate character recognition». Journal of Beijing Institute of Technology, 2005.

20. Капар, А., Гокмен, М. «Concurrent segmentation and recognition with shape-driven fast marching methods». 18-а Міжнародна конференція з розпізнавання образів (ICPR’06), Гонконг, Китай, 2006.

21. Functional and Nonfunctional Requirements: Specification and Types [Електронний ресурс] – Режим доступу до ресурсу: https://www.altexsoft.com/blog/functional-and-non-functional-requirements-specification-and-types/

22. Why Python keeps growing, explained [Електронний ресурс] – Режим доступу до ресурсу: https://github.blog/developer-skills/programming-languages-and-frameworks/why-python-keeps-growing-explained/

23. A Comprehensive Guide to Google Colab: Features, Usage, and Best

Practices [Електронний ресурс] – Режим доступу до ресурсу: https://www.analyticsvidhya.com/blog/2020/03/google-colab-machine-learning-deep-learning/

24. Software Architecture [Електронний ресурс] – Режим доступу до ресурсу: https://www.sei.cmu.edu/our-work/software-architecture/

25. Test Scenario vs Test Case [Електронний ресурс] – Режим доступу до ресурсу: https://www.practitest.com/resource-center/article/test-scenario-vs-test-case/#:~:text=What%20is%20a%20Test%20Scenario,face%20while%20using%20the%20software

26. A mojo dojo casa checklist for software testing: learn how to create [Електронний ресурс] – Режим доступу до ресурсу: https://zebrunner.com/blog-posts/a-mojo-dojo-casa-checklist-for-software-testing-learn-how-to-create

ДОДАТОК А

ВИХІДНИЙ КОД ОСНОВНИХ КОМПОНЕНТІВ

Файл ocr.py:

```
!pip install pytesseract
!sudo apt install tesseract-ocr

import cv2
import pytesseract
import matplotlib.pyplot as plt
from pylab import rcParams
from pytesseract import Output

# Параметр бібліотеки matplotlib, який задає розміри фігури (графіка)
rcParams['figure.figsize'] = 16, 8

# Завантаження зображення
file_name = "image1.jpg"
img = cv2.imread(file_name)

# Перевірка на успішне завантаження зображення
if img is None:
    print(f"Помилка: Не вдалося завантажити зображення '{file_name}'. Перевірте шлях
до файлу.")
else:
    # Попередня обробка зображення

    # Перетворення на відтінки сірого
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Зменшення шуму за допомогою Gaussian Blur
    blur = cv2.GaussianBlur(gray, (5, 5), 0)

    # Підвищення контрасту за допомогою CLAHE
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
    contrast = clahe.apply(blur)

    # Бінаризація зображення (чорно-біле перетворення)
    _, binary_img = cv2.threshold(contrast, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)

    # Виведення оригінального та обробленого зображень з підписами
    fig, axs = plt.subplots(1, 2, figsize=(16, 8))

    # Оригінальне зображення
    axs[0].imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    axs[0].set_title('Оригінальне зображення')
```

```
    axs[0].axis('off')

    # Оброблене зображення (чорно-біле)
    axs[1].imshow(binary_img, cmap='gray')
    axs[1].set_title('Оброблене чорно-біле зображення')
    axs[1].axis('off')

    plt.show()

    # Отримання інформації про текстові блоки
    try:
        data = pytesseract.image_to_data(binary_img, output_type=Output.DICT)
    except pytesseract.TesseractError as e:
        print(f"Помилка при розпізнаванні тексту: {e}")
        exit(1)

    # Виведення розпізнаного тексту
    recognized_text = " ".join(data['text']).strip()
    if recognized_text:
        print("Розпізнаний текст:")
        # Розділення тексту на рядки
        for line in recognized_text.splitlines():
            if line.strip():  # Перевірка, чи рядок не пустий
                print(line)  # Вивід кожного рядка

        # Збереження розпізнаного тексту у файл
        with open("recognized_text.txt", "w", encoding="utf-8") as text_file:
            text_file.write(recognized_text)
        print("Розпізнаний текст збережено у файл 'recognized_text.txt'.")
    else:
        print("Текст не був розпізнаний.")

    # Відображення контурів блоків тексту на оригінальному зображенні (для лівого
subplot)
    n_boxes = len(data['text'])
    img_with_boxes = img.copy()
    for i in range(n_boxes):
        if data['text'][i].strip() != "":  # Перевірка, чи блок містить текст
            (x, y, w, h) = (data['left'][i], data['top'][i], data['width'][i],
data['height'][i])
            # Малювання прямокутників навколо тексту
            img_with_boxes = cv2.rectangle(img_with_boxes, (x, y), (x + w, y + h),
(0, 255, 0), 2)

    # Відображення контурів блоків та тексту над кожним блоком (для правого subplot)
    img_with_text = img.copy()
    for i in range(n_boxes):
        if data['text'][i].strip() != "":  # Перевірка, чи блок містить текст
            (x, y, w, h) = (data['left'][i], data['top'][i], data['width'][i],
data['height'][i])
            # Малювання прямокутників навколо тексту
```

```
            img_with_text = cv2.rectangle(img_with_text, (x, y), (x + w, y + h), (0,
255, 0), 2)
            # Додавання розпізнаного тексту над прямокутником
            img_with_text = cv2.putText(img_with_text, data['text'][i], (x, y - 10),
                                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 0, 0),
2)

    # Відображення двох subplot: лівий — тільки з прямокутниками, правий — з текстом
    fig, axs = plt.subplots(1, 2, figsize=(16, 8))

    # Ліве зображення — з прямокутниками навколо блоків
    axs[0].imshow(cv2.cvtColor(img_with_boxes, cv2.COLOR_BGR2RGB))
    axs[0].set_title("Зображення з виділеними блоками")
    axs[0].axis('off')

    # Праве зображення — з прямокутниками та текстом над ними
    axs[1].imshow(cv2.cvtColor(img_with_text, cv2.COLOR_BGR2RGB))
    axs[1].set_title("Зображення з текстом над блоками")
    axs[1].axis('off')

    plt.show()

    # Збереження результатів
    cv2.imwrite("image_with_boxes.jpg", img_with_boxes)
    cv2.imwrite("image_with_text.jpg", img_with_text)
    print("Результати збережено у файлах 'image_with_boxes.jpg' та
'image_with_text.jpg'.")
```