ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ І. І. МЕЧНІКОВА

(повне найменування закладу вищої освіти)

Факультет математики, фізики та інформаційних технологій

(повне найменування факультету)

Кафедра інформаційних технологій

(повна назва кафедри)

Кваліфікаційна робота

на здобуття ступеня вищої освіти «Магістр»

«Розробка системи для оптичного розпізнавання символів (OCR) з використанням класичних методів**»**

(тема кваліфікаційної роботи українською мовою)

«Development of a system for optical character recognition (OCR) using classical methods»

(тема кваліфікаційної роботи англійською мовою)

Виконав: здобувач денної форми навчання

спеціальності 122 Комп'ютерні науки

(код, назва спеціальності)

Освітня програма Комп'ютерні науки

(назва)

Павлій Максим Андрійович

(прізвище, ім'я, по-батькові здобувача)

Керівник к.т.н., доцент Казакова Н.Ф. _____

(науковий ступінь, вчене звання, прізвище, ініціали) (підпис)

Рецензент к.т.н., доцент, доцент кафедри інформаційних технологій НУ ОЮА, Щербина Ю.В.

(науковий ступінь, вчене звання, прізвище, ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
Інформаційних технологій
№ ___ від ___ _____ 2024 р.

Завідувачка кафедри

_____ _____

(підпис) (прізвище,ім'я)

Захищено на засіданні ЕК № _____
протокол № ___ від _____ 2024 р.

Оцінка _____ / _____ / _____

(за національною шкалою/шкалою ECTS/ бали)

Голова ЕК

_____ _____

(підпис) (прізвище,ім'я)

Одеса 2024

АНОТАЦІЯ

Метою магістерської кваліфікаційної роботи є створення ефективного та надійного інструменту для автоматичного зчитування та розпізнавання тексту зображень, що дозволяє забезпечити швидку обробку та цифровізацію текстової інформації.

Методи розробки базуються на класичних методах обробки зображень для виділення та розпізнавання тексту, а також на мові програмування Python та бібліотеках Pillow, Tesseract OCR, Scikit-Image і OpenCV.

Результатом роботи є система для оптичного розпізнавання символів (OCR) з використанням класичних методів.

Ключові слова: бінаризація, морфологічні операції, методи шаблонного зіставлення, OCR-системи, OpenCV, Python, Pillow, Scikit-Image, Tesseract OCR.

ABSTRACT

The goal of this works is to create an effective and reliable tool for automatic reading and recognition of the text of images, which allows for quick processing and digitization of text information.

The development methods are based on classical image processing methods for text extraction and recognition, as well as the Python programming language and the Pillow, Tesseract OCR, Scikit-Image and OpenCV libraries.

The result of the work is a system for optical character recognition (OCR) using classical methods.

Keywords: binarization, morphological operations, pattern matching methods, OCR systems, OpenCV, Python, Pillow, Scikit-Image, Tesseract OCR.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ ТА ТЕРМІНІВ

Скорочення

NLP – Natural Language Processing

OCR – оптичне розпізнавання символів

OEM– OCR Engine Modes

OpenCV – Open Source Computer Vision Library

PIL – Python Imaging Library

PSM – Page Segmentation Modes

SSD – Sum of Squared Differences

Терміни

морфологічні операції – це набір технік, що використовуються в обробці зображень для обробки та зміни форми об'єктів на зображенні

процес бінаризації – це перетворення кольорового (або градацій сірого) зображення в двокольорове чорно-біле

template matching – методи шаблонного зіставлення

ВСТУП

Оптичне розпізнавання символів (OCR) – це спеціалізоване програмне забезпечення, призначене для розпізнавання та виділення тексту із зображень або сканованих документів. Перетворюючи фізичний текст у машинозчитувані дані, OCR дає змогу шукати, редагувати та обробляти вміст відсканованих документів за допомогою програмного забезпечення для обробки текстів. OCR є актуальним, оскільки автоматизація зчитування тексту зображень спрощує обробку документів, цифровізацію паперових архівів, роботу з рукописними чи друкованими текстами.

OCR технологія економить час, знижує ризик людських помилок і дозволяє інтегрувати текстову інформацію в різноманітні цифрові системи для подальшого пошуку, аналізу та обробки. Сьогодні OCR має широкий спектр застосувань – від мобільних сканерів і сервісів розпізнавання тексту до великих систем для автоматизації документопотоків.

Традиційно обробка банківських виписок, платіжних відомостей, рахунків-фактур або контрактів потребує годин введення даних вручну, що часто супроводжується помилками та затримками. Завдяки технології оптичного розпізнавання символів (OCR) це не тільки можливо, але й стає все більш поширеним [1].

Оптичне розпізнавання символів (OCR) стало порятунком для організацій, які прагнуть спростити роботу та підвищити ефективність. Global Data повідомляє, що до 2026 року ринок оптичного розпізнавання тексту зросте до 13,38 мільярда доларів завдяки його здатності автоматизувати вилучення даних і значно скоротити час обробки.

Розпізнавання OCR аналізує світлі та темні області зображення, щоб визначити символи тексту. Після цього він зіставляє ці символи з базою даних шаблонів, що дозволяє програмному забезпеченню розпізнавати та перетворювати їх у цифровий текст. Потім цим текстом можна маніпулювати, шукати чи архівувати відповідно до потреб користувача.

Технологія може обробляти як друкований, так і рукописний текст. Однак його точність може відрізнятися залежно від якості оригінального документа та складності програмного забезпечення OCR.

Метою роботи є створення ефективного та надійного інструменту для автоматичного зчитування та розпізнавання тексту зображень, що дозволяє забезпечити швидку обробку та цифровізацію текстової інформації. Для досягнення мети необхідно вирішити наступні задачі:

1. Оглянути методи оптичного розпізнавання символів.
2. Обрати засоби розробки OCR-системи без використання нейронної мережі.
3. Виконати аналіз вимог до системи.
4. Спроектувати архітектуру системи та описати алгоритми роботи її компонентів.
5. Після реалізації системи провести функціональне та експериментальне тестування.

Об’єкт дослідження – процес оптичного розпізнавання тексту на зображеннях, що включає етапи завантаження, попередньої обробки, розпізнавання тексту та постобробки отриманих результатів. Цей процес досліджується для підвищення точності та ефективності розпізнавання тексту.

Предмет дослідження – класичні методи обробки зображень для покращення якості тексту на вхідних зображеннях (такі як бінаризація, морфологічні операції, контрастування та нормалізація), а також алгоритми розпізнавання тексту за допомогою Tesseract OCR.

1 ОГЛЯД МЕТОДІВ ОПТИЧНОГО РОЗПІЗНАВАННЯ СИМВОЛІВ

1.1 Принципи роботи OCR-систем

OCR широко використовується для багатьох інших цілей. Процес внесення чеку у банкомат, надсилання пошти через автоматизовану систему або сканування квитанції за допомогою телефону, є прикладами переваг технології OCR. Очікується, що найближчими роками глобальний ринок оптичного розпізнавання символів швидко зростатиме. Розмір ринку OCR оцінювався в 8.93 млрд доларів у 2021 році. Очікується, що він зросте на a CAGR 15.4% між 2022 і 2030 роками. Це зростання зумовлене зростанням попиту на OCR у різних галузях кінцевого використання, таких як охорона здоров’я, автомобілебудування та інші [2].

Завдяки своїй ефективності та доступності оптичне розпізнавання символів стало основним рішенням як для професійного, так і для споживчого рівня програмного забезпечення для сканування для оцифровування документів із великою кількістю тексту та негайного надання цієї інформації (рис. 1.1).

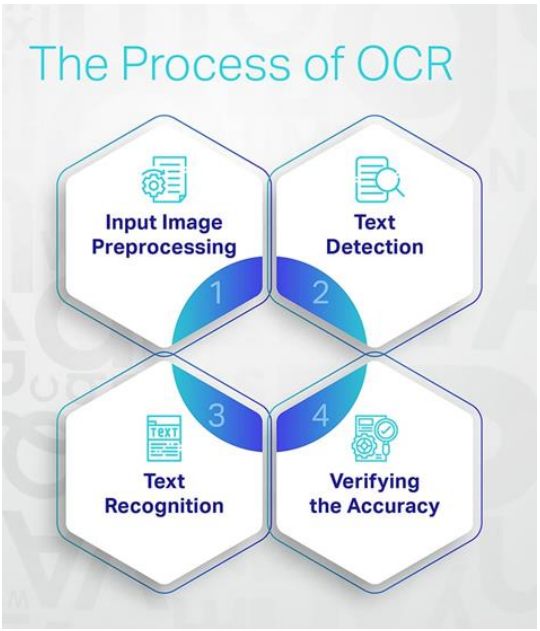


Рисунок 1.1 – Складові процесу OCR

Оптичне розпізнавання символів – це детальний процес, який допомагає витягувати текст із зображень за допомогою NLP:

1. Першим кроком у OCR є обробка вхідного зображення. Це передбачає очищення зображення та його придатність для подальшої обробки.
2. Далі механізм OCR шукає області, які містять текст на зображенні. Механізм сегментує ці області на окремі символи або слова, щоб пізніше їх можна було ідентифікувати під час розпізнавання тексту.
3. Використовуючи результати виявлення тексту, механізм OCR ідентифікує кожен символ за його формою та розміром.
4. Щойно програмне забезпечення OCR завершить розпізнавання тексту у файлі зображення, його потрібно перевірити на точність, перш ніж його можна буде використовувати [2].

Головна проблема OCR полягає в тому, що воно не ідеальне. Якщо ви уявите, що читаєте текст на цій сторінці за допомогою камери, а потім перетворюєте ці зображення на слова, ви зрозумієте, чому OCR може бути проблематичним. Деякі з проблем для OCR включають:

1. Розмитий текст, спотворений тінями.
2. Колір фону і тексту мають схожі кольори.
3. Частини зображення обрізано або повністю обрізано (наприклад, нижня частина «цього»).
4. Слабкі позначки на верхній частині деяких літер (наприклад, «і») можуть збити з пантелику програмне забезпечення оптичного розпізнавання символів і вважати їх частиною літери, а не позначки на верхній частині.
5. Різні типи та розміри шрифтів може бути важко визначити.
6. Умови освітлення під час фотографування або сканування документа.

Випадки використання:

1. Автоматизація введення даних: OCR можна використовувати для автоматизації процесу введення даних у базу даних.
2. Сканування штрих-коду: OCR дозволяє комп’ютеру сканувати штрих-коди на продуктах і отримувати інформацію про них із баз даних.
3. Розпізнавання номерних знаків: OCR аналізує номерні знаки та витягує з них таку інформацію, як реєстраційні номери та назви держав.
4. Перевірка паспорта: OCR можна використовувати для перевірки автентичності паспортів, віз та інших проїзних документів.
5. Розпізнавання етикеток магазинів: Магазини можуть використовувати оптичне розпізнавання символів, щоб автоматично зчитувати етикетки своїх продуктів і порівнювати їх із каталогами продуктів, щоб визначити, які продукти зараз є на полицях магазинів, які товари відсутні в наявності чи помилки складського приміщення.
6. Обробка страхових заяв: Програмне забезпечення OCR може сканувати документи та перевіряти підписи, дати, адреси та іншу інформацію у формах, поданих клієнтами, які подали претензії щодо збитків, завданих стихійними лихами, пожежами чи крадіжками.
7. Читання світлофора: За допомогою системи OCR можна зчитувати кольори на світлофорах і визначати, червоні вони чи зелені.
8. Зчитування лічильників комунальних послуг: Комунальні підприємства використовують OCR для зчитування лічильників електроенергії, газу та води, щоб виставляти клієнтам правильні рахунки.
9. Моніторинг соціальних медіа – Компанії використовують OCR для ідентифікації та класифікації згадок про компанію чи бренд у публікаціях у соціальних мережах, твітах і навіть оновленнях Facebook [1].

1.2 Класичні методи обробки зображень для виділення та розпізнавання тексту

Класичні методи обробки зображень для виділення та розпізнавання тексту ґрунтуються на кількох етапах, кожен із яких спрямований на покращення якості зображення та виділення ключових елементів, які полегшують процес розпізнавання тексту.

1.2.1 Бінаризація

Процес бінаризації – це перетворення кольорового (або градацій сірого) зображення в двокольорове чорно-біле. Головним параметром такого перетворення є поріг t. t – значення, з яким порівнюється яскравість кожного пікселя. За результатами порівняння пікселю присвоюється значення 0 або 1. Існують різні методи бінаризації, які умовно поділяються на дві групи: глобальні та локальні. У першому випадку поріг залишається незмінним протягом усього процесу бінаризації. У другому випадку зображення розбивається на області, і в кожній з них обчислюється свій локальний поріг.

Основною метою бінаризації є значне зменшення обсягу інформації, з якою доводиться працювати. Простіше кажучи, вдала бінаризація суттєво полегшує подальшу обробку зображення. Водночас, невдачі в процесі бінаризації можуть призвести до таких спотворень, як розриви в лініях, втрата важливих деталей, порушення цілісності об’єктів, поява шуму та непередбачувані спотворення символів через неоднорідність фону.

Різні методи бінаризації мають свої слабкі сторони: наприклад, метод Оцу може спричинити втрату дрібних деталей і "злипання" близьких символів, тоді як метод Ніблека може створювати помилкові об’єкти у випадках неоднорідного фону з низькою контрастністю. Це вказує на те, що кожен метод доцільно використовувати у відповідній області [3].

1.2.2 Морфологічні операції

Морфологічна обробка зображень намагається усунути недоліки двійкових зображень, оскільки двійкові області, утворені простим пороговим значенням, можуть бути спотворені шумом. Це також допомагає згладити зображення за допомогою операцій відкриття та закриття.

Морфологічні операції можуть бути поширені на зображення у відтінках сірого. Вона складається з нелінійних операцій, пов'язаних зі структурою ознак зображення. Це залежить від відповідного порядку пікселів, але від їх числових значень.

Морфологічні операції – це набір технік, що використовуються в обробці зображень для обробки та зміни форми об'єктів на зображенні. Вони особливо корисні для виділення та покращення структурованих елементів, як-от текст, контури символів, а також для фільтрації шуму. Основні морфологічні операції включають ерозію, дилатацію, відкриття, закриття, а також деякі інші спеціалізовані операції. Розгляньмо кожну з них більш детально [4].

1. Ерозія:
   - призначення: видалення маленьких білих точок або тонких ліній, згладжування країв об'єктів і зменшення розміру білих областей на зображенні;
   - процес: в ерозії кожен піксель зображення змінюється залежно від сусідніх пікселів, зазвичай на основі структуруючого елементу (невеликої фігури, наприклад, квадрату або кола). Піксель залишається білим, тільки якщо всі пікселі в околі структуруючого елемента також білі;
   - результат: розмір білих об’єктів зменшується, краї стають гладшими, а дрібніші деталі або шум видаляються;
   - застосування: у випадку OCR, ерозія може використовуватись для видалення шумів та тонких ліній, які не належать до основних символів.

2. Дилатація:
   - призначення: розширення розміру білих областей та злиття сусідніх об’єктів. Дилатація робить об’єкти на зображенні більшими;
   - процес: піксель залишається білим, якщо хоча б один піксель у його околі, заданому структуруючим елементом, є білим;
   - результат: білі області збільшуються, зображення виглядає «розширеним». Це дозволяє «залатати» прогалини в об’єктах і з'єднати розірвані частини тексту;
   - застосування: дилатація може допомогти, якщо текст на зображенні містить переривання чи тонкі розриви в символах, об'єднуючи ці фрагменти в єдиний символ.
3. Відкриття (Opening):
   - призначення: видалення невеликих об’єктів або шуму, що не належать до основного зображення;
   - процес: відкриття – це послідовне виконання ерозії, а потім дилатації. Спочатку видаляються дрібні білі об’єкти, а потім зберігається структура основних об'єктів;
   - результат: зображення звільняється від шуму та дрібних артефактів, а також покращується чіткість великих об'єктів. Після відкриття залишаються лише ті об'єкти, які є більшими за структуруючий елемент;
   - застосування: у випадку тексту, відкриття корисне для видалення дрібних розривів і точок, які можуть з’явитися як шум під час сканування.
4. Закриття (Closing):
   - призначення: заповнення дрібних прогалин і отворів у білих областях, що допомагає об’єднати фрагментовані частини тексту;

- процес: закриття – це послідовне виконання дилатації, а потім ерозії. Спочатку об’єкти розширюються, щоб заповнити порожні місця, а потім ерозія повертає їх до початкової форми;
- результат: після закриття зображення виглядає цільнішим, оскільки закриття допомагає об’єднати сусідні об'єкти;
- застосування: у OCR закриття часто застосовують для усунення прогалин усередині символів або між частинами одного символу, як-от між «зонтиками» літер «р», «д», «а» тощо [4].

Важливу роль у морфологічних операціях відіграє структуруючий елемент, який визначає, як саме оброблятимуться сусідні пікселі. Він може мати форму квадрата, кола, хреста або будь-якої іншої структури залежно від форми, яку потрібно обробити (рис. 1.2). Квадратний структуруючий елемент 'A' поміщається в об'єкт, який ми хочемо вибрати, 'B' перетинає об'єкт, а 'C' знаходиться поза об'єктом.

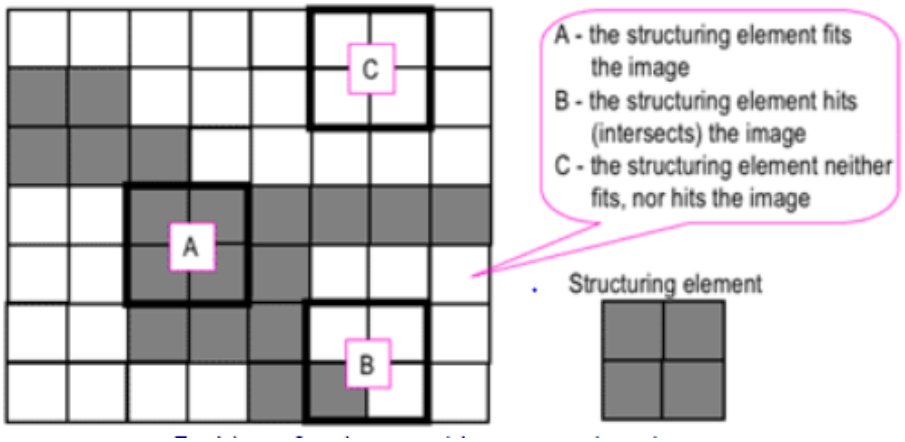


Рисунок 1.2 – Представлення структуруючого елементу у вигляді мартиці

Розмір структуруючого елемента також має значення: більші елементи виділяють крупніші структури та ігнорують дрібні деталі, тоді як менші структуруючі елементи дозволяють точніше обробити текст та зберегти більшість дрібних деталей [5].

Патерн «нуль-один» визначає конфігурацію структуруючого елемента. Це залежить від форми об'єкта, який ми хочемо вибрати (рис. 1.3). Центр структуруючого елемента ідентифікує оброблюваний піксель.

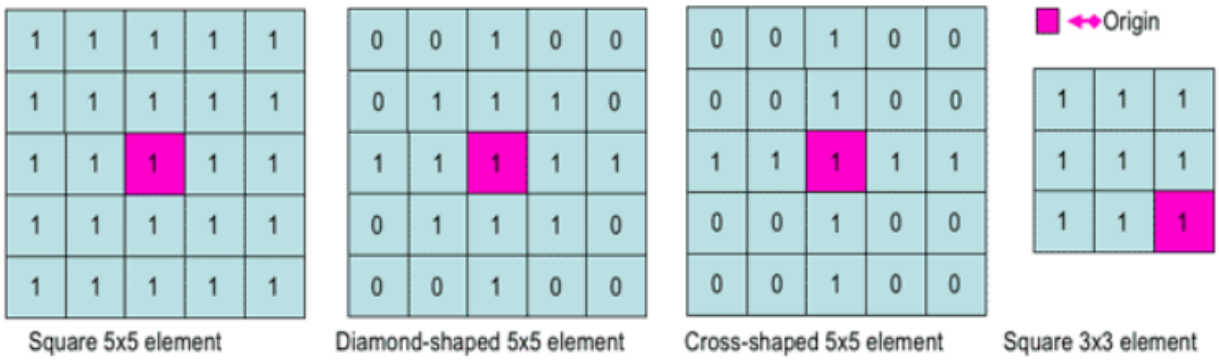


Рисунок 1.3 – Приклад патерну структуруючого елементу

Морфологічні операції є важливою складовою частиною обробки зображень у системах OCR. Вони не лише видаляють шум, але й допомагають коректно виділити символи та поліпшити їх форму перед передачею на етап класифікації [5].

1.2.3 Методи шаблонного зіставлення

Методи шаблонного зіставлення (template matching) є одними з основних класичних підходів у задачах комп'ютерного зору, зокрема в оптичному розпізнаванні символів (OCR). Ці методи базуються на зіставленні фрагментів зображення зі зразками (шаблонами), що представляють кожен символ або об’єкт, який потрібно розпізнати (рис.1.4).

В OCR шаблонне зіставлення є корисним, коли символи мають стабільну форму, шрифт і розмір, але менш ефективним у випадках із значною варіацією у вигляді тексту [6].

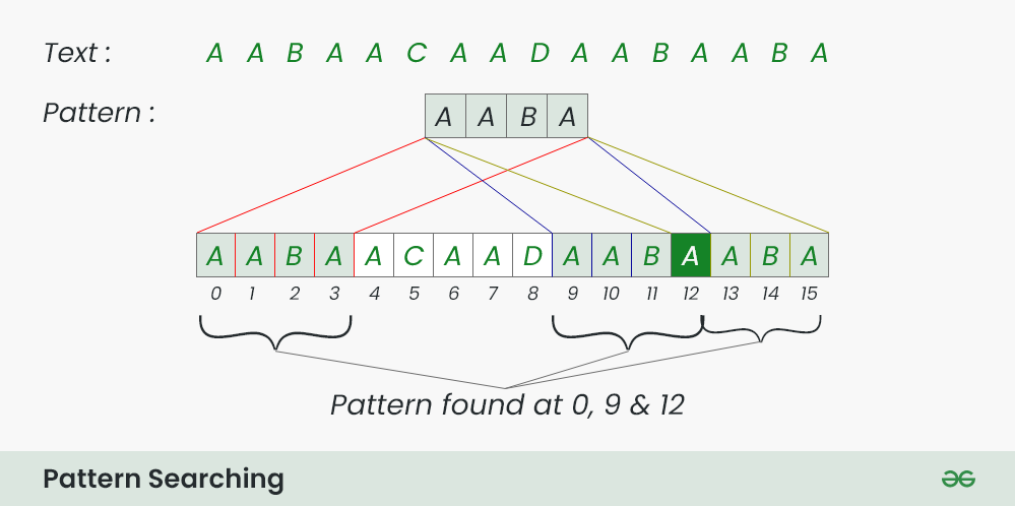


Рисунок 1.4 – Приклад використання шаблонів

Основні етапи шаблонного зіставлення:

1. Попередня обробка зображення: перед застосуванням шаблонного зіставлення необхідно підготувати зображення. Зазвичай, це включає бінаризацію, нормалізацію розміру символів, а також видалення шуму. Це полегшує процес зіставлення, оскільки шаблони також, зазвичай, є чорно-білими зображеннями.

2. Створення та зберігання шаблонів, тобто попередньо підготовлені зразки символів або об'єктів, які мають фіксовану форму, розмір і орієнтацію.

Наприклад, для кожної букви або цифри створюється шаблон, що відповідає її стандартному зображенню. Ці шаблони зберігаються у вигляді матриць пікселів або графічних файлів, що слугують еталонами для зіставлення.

3. Зіставлення шаблонів.

Сканування зображення: зображення сканується, і для кожного фрагмента перевіряється, чи є він схожим на який-небудь із шаблонів;

Віконна техніка: для зіставлення часто використовують вікно, яке "ковзає" по зображенню, порівнюючи кожен фрагмент із шаблоном. Розмір вікна повинен відповідати розмірам шаблону [6].

4. Вимірювання подібності.

Нормалізована крос-кореляція: один із найпопулярніших методів для вимірювання подібності між шаблоном і фрагментом зображення. Розраховується кореляція між шаблоном і вікном, що дає міру відповідності. Кореляційне значення вказує, наскільки фрагмент відповідає шаблону, де значення ближчі до 1 означають високу подібність;

Метрика відмінності пікселів: різниця між інтенсивностями пікселів у шаблоні й фрагменті зображення. Якщо сума відмінностей менша за певний поріг, вважається, що шаблон знайдено.

Метод сум квадратичних відхилень (SSD – Sum of Squared Differences): розрахунок сум квадратів різниць між пікселями шаблона та відповідного вікна зображення. Чим менше значення SSD, тим краще підходить шаблон.

5. Прийняття рішення.

Після обчислення подібності між шаблоном і фрагментом зображення приймається рішення про те, який символ (або інший об'єкт) найкраще відповідає знайденому фрагменту. Якщо зіставлення не досягає заданого порогу, це може свідчити про те, що відповідного шаблону на зображенні немає, або зображення символу спотворене [5].

Переваги шаблонного зіставлення:

1. Простота: метод є досить простим у реалізації, оскільки не вимагає складних обчислень або попереднього навчання.
2. Точність для стандартних шрифтів: ефективний у випадках, коли текст має чіткий та однаковий шрифт, орієнтацію та розмір.
3. Висока швидкість на малих обсягах даних: працює швидко, якщо кількість шаблонів не є великою.

Недоліки шаблонного зіставлення:

1. Чутливість до змін у тексті: якщо текст змінює розмір, шрифт або орієнтацію, метод стає неефективним, оскільки шаблони не співпадають із варіативними формами символів.

2. Чутливість до шуму: шум і спотворення зображення значно знижують точність шаблонного зіставлення.
3. Не підходить для великих бібліотек: якщо кількість символів або їх варіантів є великою, цей метод стає ресурсомістким, оскільки вимагає зіставлення з великою кількістю шаблонів.

Шаблонне зіставлення використовується в OCR для розпізнавання друкованого тексту в строго визначених форматах, таких як ідентифікаційні номери або коди, де символи завжди однакові. Крім цього, шаблонне зіставлення знаходить своє використання у розпізнаванні спеціальних символів у документах, штрих-кодах, QR-кодах або для перевірки підписів, коли об’єкти мають відносно сталі форми [7].

Методи шаблонного зіставлення є корисними для конкретних завдань і формують важливий базовий підхід для OCR. Водночас для більш варіативних та складних сценаріїв у сучасних додатках їх поступово заміняють нейронні мережі та інші алгоритми машинного навчання, які можуть працювати з ширшими варіаціями тексту й шуму.

1.2.4 Розтягнення контрасту та нормалізація

Розтягнення контрасту та нормалізація є важливими етапами попередньої обробки в OCR, оскільки ці методи допомагають покращити чіткість тексту та підвищити контраст між символами і фоном. Це особливо корисно для зображень із слабким освітленням або низькою контрастністю, де текст зливається з фоном, і для забезпечення максимальної точності розпізнавання.

Розтягнення контрасту дозволяє підсилити видимість слабоконтрастного тексту, роблячи його чіткішим щодо фону:

1. Аналіз початкового діапазону інтенсивностей: спочатку обчислюється діапазон значень інтенсивності пікселів на зображенні (від найтемнішого до найсвітлішого).

2. Розширення діапазону: потім цей діапазон розширюється до максимально можливого (наприклад, 0–255 для 8-бітних зображень), розтягуючи інтенсивності в кожному пікселі.
3. Перетворення інтенсивності: пікселі з низькою інтенсивністю темнішають, а з високою – освітлюються, що створює контраст між текстом і фоном.

Переваги такого підходу наступні:

1. Покращує видимість тонких деталей на зображенні.
2. Підвищує чіткість тексту для подальшої обробки, наприклад, порогування чи сегментації.

Нормалізація є методикою, яка змінює яскравість і контрастність зображення, застосовуючи математичне вирівнювання інтенсивності пікселів. На відміну від простого розтягнення контрасту, нормалізація може використовувати різні підходи, наприклад, видалення аномальних значень або рівномірне розподілення інтенсивності за середнім значенням [6].

Алгоритм нормалізації наступний:

1. Розрахунок середнього та стандартного відхилення: обчислюються середнє значення та стандартне відхилення інтенсивностей, щоб скоригувати загальну яскравість.
2. Зміна значень інтенсивності: кожен піксель нормалізується так, щоб яскравість кожного з них відповідала певному діапазону.
3. Розподіл рівнів яскравості: цей підхід особливо корисний для зображень із нерівномірним освітленням, оскільки нормалізація згладжує інтенсивності, вирівнюючи кольори фону та тексту.

1.2.5 Розтягнення гістограми

Мета методу розтягнення гістограми – це розширити діапазон інтенсивностей зображення, щоб збільшити контрастність між темними (символами) і світлими (фоном) частинами зображення.

Принцип дії наступний: розтягнення гістограми виконує лінійне перетворення інтенсивностей пікселів, що дозволяє найтемніші значення наблизити до чорного (0), а найсвітліші – до білого (255 для 8-бітного зображення). Це зменшує кількість проміжних відтінків і робить межі між об'єктами на зображенні більш виразними.

Розтягнення гістограми працює наступним чином:

1. Визначаються мінімальні та максимальні значення інтенсивності на зображенні.
2. Далі ці значення нормалізуються, тобто перетворюються так, щоб покрити весь діапазон інтенсивностей.

Формула для кожного пікселя при лінійному розтягненні гістограми виглядає так:

$$I_{new} = \frac{(I_{old} - I_{\min})}{(I_{max} - I_{\min})} x\ 255 \quad (1.1)$$

I_{old} – початкова інтенсивність пікселя, $I_{\min}$, I_{max} – мінімальні та максимальні значення інтенсивності на зображенні, а 255 – верхня межа для 8-бітного зображення.

Такий підхід дозволяє "розтягнути" початкові значення інтенсивності по всьому діапазону, що робить затемнені символи яскравішими і чіткішими на фоні. Якщо символи на оригінальному зображенні виглядають сірими через погане освітлення, після розтягнення гістограми темні пікселі стануть чорнішими, а світлі – яскравішими. Це створює чіткий контраст між текстом і фоном, покращуючи візуальну видимість тексту [7].

Метою підходу гістограмного вирівнювання є перерозподіл значень інтенсивності так, щоб вони були більш рівномірно розподілені по всьому діапазону. Це дозволяє поліпшити контрастність на зображеннях із вузьким діапазоном яскравості, де всі інтенсивності пікселів можуть бути близькими до одного значення.

Гістограмне вирівнювання не просто розтягує інтенсивності, а змінює їхній розподіл таким чином, щоб уникнути накопичення пікселів в одній частині діапазону. Воно збільшує контраст між пікселями, що мають подібні інтенсивності, розширюючи ці значення по всьому діапазону.

Як працює гістограмне вирівнювання:

1. Спочатку будується гістограма інтенсивностей пікселів для початкового зображення.
2. Потім для кожного значення інтенсивності розраховується його кумулятивна частота (сума частот усіх менших значень інтенсивності). Це дозволяє відобразити інтенсивності в новий діапазон, зберігаючи відносний порядок яскравості пікселів.

Для 8-бітного зображення, де інтенсивності варіюються від 0 до 255, формула вирівнювання виглядає так:

$$I_{new} = CDF(I_{old}) * 255 \quad (1.2)$$

де CDF – кумулятивна частотна функція.

Результат – зображення, в якому темніші та світліші області стали чіткіше помітні, а відтінки, що раніше зливались, стали більш контрастними.

Приклад застосування: якщо на оригінальному зображенні всі символи та фон мають подібні значення інтенсивності (наприклад, через погане освітлення), після вирівнювання гістограми яскравість пікселів буде перерозподілена, і текст стане чіткішим. Це також ефективно при обробці відсканованих документів, де через нерівномірне освітлення та затінення текст і фон можуть мати близькі значення яскравості.

1.3 Обґрунтування вибору класичних методів обробки зображень без використання нейронних мереж

Вибір класичних методів обробки зображень для задачі оптичного розпізнавання символів (OCR) без використання нейронних мереж може бути обґрунтований кількома важливими факторами, які стосуються ефективності, вартості обчислень, простоти реалізації та специфічних вимог проекту.

1. Обмежені обчислювальні ресурси.

Мала кількість пам'яті: класичні методи часто є менш вимогливими до пам’яті, оскільки не вимагають зберігання великих моделей чи складних матриць, які є типовими для нейронних мереж.

Мала обчислювальна потужність: відсутність нейронних мереж значно знижує обчислювальні витрати, адже класичні алгоритми не потребують численних циклів оптимізації, які є необхідними для навчання моделей глибокого навчання. Це є корисним для вбудованих систем або пристроїв із низькою продуктивністю.

2. Висока швидкість і простота реалізації.

Прямий підхід: класичні методи обробки зображень, такі як порогове значення, морфологічні операції, контурне виділення та шаблонне зіставлення, є простими у реалізації та не потребують тривалого тренування, яке займає час і ресурси.

Низька затримка: у випадках, коли потрібне швидке розпізнавання в реальному часі, класичні методи можуть бути ефективнішими, адже вони не обтяжені складними обчисленнями, що характерно для глибинних мереж.

3. Чітко визначені умови для розпізнавання.

Стандартні шрифти та фіксовані розміри: якщо завдання полягає в розпізнаванні символів одного типу (наприклад, стандартних друкованих шрифтів), класичні методи, такі як шаблонне зіставлення, є надзвичайно ефективними і можуть забезпечити високу точність без потреби в складних моделях [8].

Відсутність складної варіативності: у випадках, коли текст не має варіативності у вигляді шрифту, нахилу, або шуми можна легко фільтрувати, класичні підходи цілком справляються з задачами OCR.

4. Легкість налаштування та модифікації.

Простота налаштувань: багато класичних методів мають лише декілька параметрів для налаштування (розмір структуруючого елементу для морфологічних операцій або порогове значення для бінаризації), що полегшує підбір параметрів та швидке налаштування системи під конкретні умови.

Гнучкість адаптації: у випадку потреби, класичні методи можна швидко адаптувати до різних умов (наприклад, зміни освітлення або фону) за рахунок налаштування порогів або інших параметрів, що є менш трудомістким у порівнянні з перенавчанням нейронних мереж.

5. Прозорість та зрозумілість алгоритмів.

Легкість інтерпретації результатів: класичні методи обробки зображень є більш прозорими та зрозумілими. Наприклад, у методах шаблонного зіставлення чітко видно, чому конкретний символ був розпізнаний певним чином, тоді як робота нейронних мереж часто виглядає як «чорний ящик».

Контрольованість процесу обробки: маючи прямий доступ до алгоритмів і можливість безпосередньо керувати їхньою логікою, легше контролювати та пояснювати результати розпізнавання. Це особливо важливо для чутливих систем, де важлива зрозумілість процесів [8].

6. Мінімізація обсягу даних для навчання.

Відсутність потреби в об'ємних наборах даних: класичні методи обробки зображень не потребують попереднього навчання, тобто немає необхідності збирати та обробляти великі обсяги даних. Це є значною перевагою у випадках, коли немає доступу до достатньої кількості навчальних даних.

Ефективність у вузькоцільових завданнях: класичні методи чудово справляються з завданнями розпізнавання, якщо кількість варіацій обмежена (наприклад, лише латинський алфавіт чи цифри), тоді як нейронні мережі

можуть вимагати значної кількості прикладів для точного навчання на кожному класі.

7. Менша вразливість до проблем з узагальненням.

Мінімізація помилок через оверфіттинг: класичні методи є менш схильними до перенавчання, оскільки вони не використовують параметричні моделі, що оптимізуються під конкретні дані. Відсутність потреби в моделюванні допомагає зберігати точність при застосуванні до нових зображень.

Стабільність при обмежених варіаціях: для завдань, де потрібно обробляти лише відомі символи в конкретних умовах (наприклад, розпізнавання цифр на лічильнику), класичні методи забезпечують високу стабільність, адже не потребують узагальнення на різні типи даних [8].

Варто зазначити, що класичні методи мають обмеження у варіативних завданнях, де шрифти, стилі символів, розміри та фони значно змінюються. У таких випадках нейронні мережі можуть бути кращими через здатність до навчання на численних варіаціях. Однак у задачах, де характеристики тексту є стабільними і не потребують складного узагальнення, класичні методи забезпечують точність та простоту, що робить їх чудовим вибором для багатьох OCR-систем.

2 ОГЛЯД ТА ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ ТА ІНСТРУМЕНТІВ

2.1 Бібліотека Pillow (PIL)

Pillow – це потужна бібліотека для обробки зображень у Python, що є оновленою версією бібліотеки PIL (Python Imaging Library). Він підтримує широкий спектр форматів зображень, таких як PPM, JPEG, TIFF, GIF, PNG і BMP. PIL була популярною бібліотекою, але припинила розвиток, і тепер її функції підтримуються та розширюються в бібліотеці Pillow. Вона надає інструменти для різноманітних операцій над зображеннями, включаючи відкриття, зміну, збереження та обробку зображень у різних форматах. Pillow має простий та інтуїтивний інтерфейс, що робить її ідеальною для базової та середньої обробки зображень, включаючи задачі підготовки для OCR [9].

Ключові функції та можливості Pillow:

1. Завантаження та збереження зображень.

Pillow підтримує велику кількість форматів зображень, зокрема PNG, JPEG, BMP, GIF, TIFF та інші. Завантажене зображення представлено у вигляді об'єкта класу «Image», що дозволяє легко працювати з ним: змінювати, зберігати у потрібному форматі або обробляти.

```
from PIL import Image
image = Image.open("example.jpg")  # Завантаження зображення
image.save("output.png")           # Збереження у форматі PNG
```

2. Перетворення форматів.

Pillow дозволяє зручно конвертувати зображення між різними форматами. Це особливо корисно для приведення всіх зображень до одного формату перед обробкою. Наприклад, можна легко конвертувати зображення із BMP в JPEG або PNG.

3. Зміна розмірів та масштабування. Pillow дозволяє змінювати розміри зображення з урахуванням пропорцій (або без них), що може бути

корисно для підготовки зображень перед обробкою. Методи resize() і thumbnail() дозволяють точно налаштувати розмір зображення.

```
    resized_image = image.resize((200, 200))# Масштабування до 200x200
    image.thumbnail((100, 100))# Пропорційне зменшення розмірів
```

4. Обертання, віддзеркалення та обрізання зображень.

Pillow має функції для обертання (rotate), віддзеркалення (transpose), а також обрізання (crop) зображень. Це корисно для підготовки зображень, які могли бути відскановані під кутом або містять зайві області.

```
    rotated_image = image.rotate(45) # Обертання на 45 градусів
cropped_image = image.crop((50, 50, 200, 200)) # Обрізання частини зображення
```

5. Бінаризація та конвертація в градації сірого.

Для задач OCR часто потрібно перевести зображення в градації сірого або чорно-білий формат. Pillow підтримує перетворення зображень у різні кольорові моделі, такі як RGB, RGBA, та L (градації сірого). Бінаризацію (переведення у чорно-білий формат) можна виконати за допомогою порогового значення.

```
    grayscale_image = image.convert("L")    # Перетворення у градації сірого
    binary_image = grayscale_image.point(lambda x: 0 if x < 128 else 255, '1')  # Бінаризація
```

6. Налаштування яскравості, контрастності та кольору

Бібліотека дозволяє змінювати яскравість, контрастність, різкість і насиченість зображення, що може допомогти у випадках, коли текст на зображенні погано видно. Для цього можна використовувати модуль «ImageEnhance», що містить інструменти для коригування різних параметрів зображення.

```
from PIL import ImageEnhance
enhancer = ImageEnhance.Contrast(image)
enhanced_image = enhancer.enhance(2)#Збільшення контрастності
```

7. Робота з гістограмами.

Pillow дозволяє отримати гістограму зображення, яка показує розподіл інтенсивностей пікселів. Це може бути корисно для аналізу контрастності зображення та виконання операцій на основі гістограми, наприклад, розтягнення або вирівнювання контрасту.

```
histogram = image.histogram()
```

8. Накладання тексту та інших зображень.

Pillow підтримує накладання тексту, що корисно для створення водяних знаків, анотацій або маркування частин зображення. Використовуючи модуль «ImageDraw», можна додавати текст із можливістю вибору шрифтів та кольорів.

```
from PIL import ImageDraw, ImageFont
draw = ImageDraw.Draw(image)
font = ImageFont.truetype("arial.ttf", 36)
draw.text((50, 50), "Example Text", font=font, fill="black")
```

9. Фільтрація та зменшення шуму.

Pillow надає доступ до фільтрів, що можуть бути корисні для згладжування або підвищення різкості зображень. Це дозволяє зменшити шум перед розпізнаванням тексту. Фільтри, такі як BLUR, SHARPEN, EDGE_ENHANCE, можна використовувати для покращення якості зображення [9].

```
from PIL import ImageFilter blurred_image =
image.filter(ImageFilter.BLUR) # Розмиття зображення
```

10. Конвертація зображень у масиви та інтеграція з NumPy.

Pillow дозволяє перетворювати зображення у масиви NumPy, що полегшує використання зображень у обчисленнях, застосування додаткових алгоритмів обробки та маніпуляції з пікселями. Це особливо зручно для використання Pillow спільно з іншими бібліотеками, такими як OpenCV та SciPy [9].

Переваги Pillow для класичного OCR:

1. Універсальність: підтримка широкого спектру форматів зображень та операцій з ними, від базових (збереження, зміна розміру) до складніших (фільтрація, обробка гістограм).
2. Зручність інтеграції: Pillow можна легко комбінувати з іншими бібліотеками, такими як OpenCV та NumPy, для створення більш складних алгоритмів обробки зображень.
3. Простота використання: інтуїтивний інтерфейс та прості команди дозволяють швидко освоїти основні операції, навіть якщо розробник не має великого досвіду в обробці зображень.
4. Ефективність для базових завдань OCR: підготовка зображення за допомогою Pillow (контраст, бінаризація, фільтрація) суттєво підвищує якість розпізнавання тексту при використанні класичних методів OCR.

Бібліотека Pillow є надійним вибором для задач передобробки зображень, особливо у випадках, коли потрібно підготувати зображення для OCR, використовуючи класичні методи без нейронних мереж.

2.2 Tesseract OCR

Tesseract OCR – це один із найпотужніших і популярних інструментів для оптичного розпізнавання тексту, який використовує алгоритми комп’ютерного бачення для витягування тексту з зображень. Спочатку Tesseract був розроблений компанією HP у 1985 році, а з 2006 року його

розвиток підтримує Google. Він є відкритим та безкоштовним інструментом, що підтримує різні мови та стилі тексту.

Ключові особливості Tesseract OCR:

1. Розпізнавання тексту на різних мовах.

Tesseract підтримує понад 100 мов, і для кожної є можливість завантажити відповідний мовний пакет. Підтримує як латиницю, так і нелатинські алфавіти, зокрема арабську, китайську, кирилицю тощо. Користувач може вибрати одну або кілька мов для розпізнавання тексту на зображенні.

```
    # Розпізнання англійського тексту
    import pytesseract from PIL import Image text =
pytesseract.image_to_string(Image.open('example.jpg'),
lang='eng')
```

2. Аналіз структури тексту

Tesseract може визначати структуру тексту на зображенні: вирізняє абзаци, заголовки, таблиці, колонки та навіть може розпізнати поля форм. Це робить його універсальним для роботи з документами, книгами, рахунками тощо.

3. Підтримка попередньої обробки зображень

Tesseract дозволяє використовувати попередню обробку зображень для підвищення точності розпізнавання. Наприклад, перед OCR можна застосувати бінаризацію, зміну контрасту, згладжування. Найчастіше Tesseract комбінують із бібліотеками для обробки зображень, таких як OpenCV або Pillow, для покращення якості зображення перед розпізнаванням [10].

```
    import cv2
    import pytesseract

    image = cv2.imread('example.jpg')
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Перетворення у градації сірого
text = pytesseract.image_to_string(gray)
```

4. Розпізнавання вбудованих символів та малих шрифтів.

Tesseract може розпізнавати текст у складних зображеннях, які містять невеликі шрифти, зображення з низькою якістю, і навіть у випадках, коли текст містить багато графічних елементів. Розпізнавання тексту в невеликих шрифтах може бути налаштоване за допомогою параметрів PSM (Page Segmentation Modes) та OEM (OCR Engine Modes), що дозволяє адаптувати поведінку OCR для конкретних завдань.

5. Налаштування режиму сегментації сторінки (PSM).

Tesseract має різні режими сегментації сторінки, які визначають, як саме OCR оброблятиме зображення: як один блок тексту, як абзаци, рядки, або навіть як окремі слова чи символи. Це особливо корисно для різних видів документів: газет, сторінок книг, або квитанцій, де текст може бути структурований по-різному.

6. Підтримка вихідного формату hOCR та ALTO XML.

Tesseract може виводити результати OCR у форматі hOCR або ALTO XML, що містить інформацію про позицію кожного розпізнаного символу. Це корисно для збереження структури документа та точного розташування тексту на сторінці. Використання hOCR-формату дозволяє інтегрувати розпізнаний текст з вихідним зображенням, щоб зберегти його оформлення.

```
    text_hocr = pytesseract.image_to_pdf_or_hocr(gray,
extension='hocr')
```

7. Можливість тренування для покращення точності.

Tesseract дозволяє користувачам тренувати власні моделі на основі власних наборів даних, що дає можливість покращити точність для специфічних шрифтів, символів або мов. Це може бути корисно, якщо потрібно розпізнати спеціалізовані документи, наприклад, з нестандартними шрифтами чи рукописним текстом [11].

8. Інтеграція з Python через «pytesseract».

Пакет «pytesseract» дозволяє використовувати Tesseract OCR безпосередньо у Python, що спрощує обробку зображень та автоматизує розпізнавання тексту. «pytesseract» може використовуватись у поєднанні з бібліотеками обробки зображень, такими як Pillow та OpenCV, для створення комплексних рішень з обробки та аналізу тексту.

Переваги використання Tesseract OCR для класичних завдань OCR:

1. Відкритий код і безкоштовність: Tesseract є повністю безкоштовним та відкритим, що робить його доступним для різних завдань, включаючи комерційні проекти.
2. Широкі можливості налаштування, а саме різні режими сегментації сторінок, мовні модулі та можливість тренування нових моделей дозволяють налаштовувати Tesseract під конкретні потреби.
3. Гнучкість при інтеграції завдяки сумісності з популярними бібліотеками, такими як OpenCV, Tesseract легко інтегрується у більш складні системи обробки зображень.

Для інтеграції Tesseract OCR з C# можна використовувати обгортку Tesseract OCR для .NET під назвою Tesseract.NET. Це популярний порт Tesseract, адаптований для роботи з C# і дозволяє легко викликати функції Tesseract OCR у середовищі .NET [11].

Основні характеристики Tesseract.NET:

1. Підтримка багатьох мов: Tesseract.NET підтримує всі ті ж мовні модулі, що й Tesseract OCR, що дозволяє розпізнавати текст на багатьох мовах, включаючи складніші алфавіти, як-от арабська, китайська, японська.
2. Просте використання з C#: обгортка надає зручний API для роботи з зображеннями, що робить її ідеальною для розробників, які працюють у середовищі .NET.
3. Легка інтеграція з обробкою зображень: Tesseract.NET можна використовувати з іншими бібліотеками, наприклад, OpenCV або

System.Drawing у C# для підготовки зображень перед розпізнаванням, що значно покращує результати OCR.

4. Можливість налаштування розпізнавання: як і оригінальний Tesseract, обгортка підтримує параметри налаштування режиму сегментації сторінок (PSM) та режиму розпізнавання (OEM), що допомагає адаптувати розпізнавання для різних типів документів.

Особливості та переваги Tesseract.NET:

- відкрите програмне забезпечення: Tesseract.NET є безкоштовною обгорткою для Tesseract OCR, і його можна використовувати у комерційних проектах;
- простий API: інтеграція Tesseract у C# досить проста, що дозволяє використовувати OCR навіть для новачків;
- підтримка багатомовності: підтримує розпізнавання багатьох мов і навіть комбінації кількох мов.

Таким чином, Tesseract.NET є зручним інструментом для розробників .NET, які хочуть додати функцію розпізнавання тексту до своїх програм без складної інтеграції.

2.3 Бібліотека Python Scikit-Image

Scikit-Image (або просто skimage) – це потужна бібліотека Python для обробки зображень, яка є частиною екосистеми Scipy. Вона надає набір інструментів для роботи з цифровими зображеннями, дозволяючи виконувати складні операції обробки зображень, сегментацію, фільтрацію та аналіз зображень з мінімальним обсягом коду. Skimage особливо популярна для наукових досліджень і часто використовується в поєднанні з бібліотеками NumPy, SciPy та Matplotlib.

Основні можливості Scikit-Image:

1. Базові операції обробки зображень.

Scikit-Image дозволяє виконувати основні операції обробки зображень, такі як масштабування, обертання, обрізання та зміна формату. Надає можливості для попередньої обробки зображень, наприклад нормалізація інтенсивностей пікселів, корекція контрасту, покращення якості, зменшення шуму та фільтрація [12].

```
from skimage import io, color

image = io.imread("example.jpg")#Завантаження зображення
gray_image = color.rgb2gray(image)#Перетворення у відтінки сірого
```

2. Фільтрація та згладжування.

Scikit-Image містить низку фільтрів для згладжування та зменшення шуму, таких як гаусівський фільтр, медіанний фільтр, фільтр Собеля для виділення країв, а також фільтри Лапласа та Кенні. Ці фільтри особливо корисні для підвищення чіткості тексту перед його розпізнаванням, зокрема для виділення контурів символів.

```
from skimage.filters import sobel
edges = sobel(gray_image)# Застосування фільтра Собеля для виявлення країв
```

3. Сегментація зображень.

Сегментація дозволяє розділити зображення на окремі області, що може бути корисним для виділення тексту або інших об’єктів. Scikit-Image підтримує такі методи сегментації, як розділення за порогом, сегментація за кластеризацією, розширення областей та методи графічної сегментації.

```
from skimage.filters import threshold_otsu

threshold = threshold_otsu(gray_image)
binary_image = gray_image > threshold  # Бінаризація зображення на основі порогу Оцу
```

4. Морфологічні операції.

Морфологічні операції в Scikit-Image дозволяють виконувати такі дії, як дилатація, ерозія, відкриття та закриття. Ці операції застосовуються для обробки бінарних зображень, зокрема для очищення шуму та покращення якості виділених об’єктів. Морфологічні операції є корисними для підготовки тексту до OCR, оскільки вони допомагають відокремити символи та видалити невеликі артефакти.

```
from skimage.morphology import binary_dilation

dilated_image = binary_dilation(binary_image)  # 
Застосування дилатації
```

5. Аналіз властивостей зображення.

Scikit-Image дозволяє вимірювати властивості зображень та окремих сегментів, як-от площа, периметр, інтенсивність та форма об’єктів. Це може бути корисно для задач аналізу зображень, коли потрібно розпізнавати певні структури чи символи на основі їх характеристик.

```
from skimage.measure import label, regionprops

labeled_image = label(binary_image)
for region in regionprops(labeled_image):
    print("Центр мас:", region.centroid)  # Визначення 
центру об'єкта
```

6. Покращення контрасту та нормалізація.

Scikit-Image містить функції для покращення контрасту, такі як еквалізація гістограми, адаптивна еквалізація та розтягнення контрасту. Нормалізація допомагає зробити зображення більш чітким та покращити видимість тексту, особливо на зображеннях зі слабким контрастом.

```
from skimage import exposure
```

```
    equalized_image = exposure.equalize_hist(gray_image)  # 
Гістограмне вирівнювання
```

7. Реалізація специфічних алгоритмів обробки зображень.

Scikit-Image надає низку алгоритмів для специфічних задач, таких як злиття зображень, аналіз текстур, виявлення шаблонів та інших методів для покращення якості зображень перед OCR.

Переваги використання Scikit-Image:

1. Відкрите програмне забезпечення та легка інтеграція: Scikit-Image є безкоштовною та відкритою бібліотекою, яка легко інтегрується з іншими Python-бібліотеками.
2. Розширені можливості обробки зображень: Scikit-Image має великий набір функцій для обробки та аналізу зображень, що робить її універсальною для багатьох задач.
3. Підтримка наукових досліджень: завдяки функціональності й простоті використання, Scikit-Image широко застосовується в наукових дослідженнях та обробці зображень в академічних проєктах [12].

2.4 Бібліотека OpenCV

OpenCV (Open Source Computer Vision Library) – це відкрита бібліотека для комп’ютерного зору та обробки зображень, яка забезпечує широкі можливості для роботи з візуальними даними. OpenCV підтримує безліч мов програмування (Python, C++, Java) та платформ (Windows, Linux, macOS, Android, iOS), і є однією з найпопулярніших бібліотек у галузі комп’ютерного зору завдяки своїй функціональності та продуктивності. OpenCV широко застосовується як у наукових дослідженнях, так і в комерційних проєктах.

Основні можливості OpenCV:

1. Обробка зображень.

OpenCV дозволяє виконувати базові операції зображень, як-от масштабування, обрізання, поворот, розмиття, бінаризація, перетворення кольорів (наприклад, перетворення RGB в градації сірого), нормалізація контрасту та ін.

Такі операції є корисними для попередньої обробки зображень, особливо перед розпізнаванням тексту, щоб зробити текст більш чітким і легким для OCR [13].

```
import cv2

image = cv2.imread('example.jpg')  # Завантаження зображення
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)    # Перетворення у відтінки сірого
```

2. Фільтрація та видалення шуму.

OpenCV містить широкий набір фільтрів для обробки зображень, таких як гаусівський, медіанний та біквадратний фільтри, які можуть зменшувати шум та покращувати якість зображень. Також є можливість виділення країв за допомогою таких фільтрів, як Собеля, Лапласа та Кенні, що корисно для підвищення чіткості тексту.

```
blurred_image = cv2.GaussianBlur(gray_image, (5, 5), 0)   # Гаусівське розмиття
edges = cv2.Canny(blurred_image, 100, 200)  # Виділення країв методом Кенні
```

3. Морфологічні операції.

OpenCV надає набір морфологічних операцій (ерозія, дилатація, відкриття, закриття), які застосовуються до бінарних зображень для видалення шуму та покращення форми об’єктів. Ці операції можуть бути особливо корисними для покращення вигляду тексту, видалення маленьких артефактів або злиття фрагментів символів.

```
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
dilated_image = cv2.dilate(binary_image, kernel, iterations=1)
# Дилатація
```

4. Сегментація та виділення об’єктів.

OpenCV пропонує різноманітні алгоритми сегментації, такі як метод порогів, адаптивне порогування, кластеризація K-середніх та методи на основі контурів. Сегментація є важливою для виділення тексту або інших об’єктів на зображенні. Адаптивне порогування дозволяє виділяти текст навіть на нерівномірно освітлених зображеннях.

5. Розпізнавання тексту та оптичне розпізнавання символів (OCR).

Хоча OpenCV не має вбудованого OCR, його функції обробки зображень часто використовуються для попередньої обробки перед OCR. Завдяки OpenCV можна підготувати зображення для використання з Tesseract OCR, виділяючи текст, коригуючи контраст і позбуваючись шуму.

6. Задачі з комп’ютерного зору.

OpenCV дозволяє розпізнавати об’єкти та осіб за допомогою методів машинного навчання, таких як Haar Cascades та DNN (глибокі нейронні мережі). Містить функції для відстеження об'єктів, виявлення руху та аналізу руху, що робить його корисним для більш складних задач комп’ютерного зору.

7. Вбудовані алгоритми для машинного навчання.

OpenCV також підтримує базові алгоритми машинного навчання, такі як SVM, K-ближчих сусідів, K-середніх, що дозволяє проводити класифікацію і кластеризацію даних зображень.

Переваги OpenCV:

- висока продуктивність: OpenCV оптимізовано для швидкої обробки зображень, що дозволяє використовувати його у реальному часі;
- різноманітність функцій: бібліотека охоплює широкий спектр задач комп’ютерного зору, обробки зображень і навіть машинного навчання;

- підтримка багатьох мов і платформ: це робить OpenCV універсальним вибором для розробників на різних платформах;
- сумісність з іншими бібліотеками: OpenCV легко інтегрується з NumPy, Pandas, TensorFlow та іншими бібліотеками для побудови комплексних рішень.

OpenCV є потужною та гнучкою бібліотекою для обробки зображень і комп'ютерного зору, яка підходить для широкого спектра проєктів, включаючи розпізнавання тексту та аналіз зображень.

2.5 Вибір засобів розробки системи

У другому розділі описано порівняний аналіз інструментів розробки системи для оптичного розпізнавання символів (OCR) з використанням класичних методів без використання нейронної мережі. Прийнято рішення використовувати описані бібліотеки в такій послідовності:

1. Завантаження зображення: використати Pillow для завантаження та попередньої конвертації зображення, якщо це потрібно.
2. Попередня обробка зображення: застосувати OpenCV для бінаризації, покращення контрасту та видалення шуму. Scikit-Image може використовуватися для розширених методів обробки, якщо якість зображення низька.
3. Розпізнавання символів: після підготовки зображення скористатися Tesseract OCR для розпізнавання тексту.

Мова програмування для реалізації обрана Python. OpenCV та Tesseract OCR – це основні інструменти для такого проєкту, причому OpenCV відповідає за обробку зображення, а Tesseract OCR – за розпізнавання тексту.

Scikit-Image та Pillow використовуються для додаткової обробки та завантаження/конвертації, якщо в цьому є потреба.

Комбінація цих бібліотек дозволить побудувати ефективну систему OCR на основі класичних методів, без залучення нейронних мереж.

3 ПРОЕКТУВАННЯ OCR-СИСТЕМИ

3.1 Аналіз вимог до системи

Аналіз вимог до системи є важливим етапом розробки, який забезпечує чітке розуміння цілей, функцій і обмежень системи. По-перше, це забезпечення відповідності очікуванням замовника, а саме аналіз вимог дозволяє зрозуміти потреби та очікування замовника або користувача, щоб створити продукт, який вирішує його завдання. Це знижує ризик розробки непотрібного або невідповідного функціоналу.

Запобігання помилкам і невизначеностям, тобто вимоги, визначені на початку, допомагають уникнути неправильних припущень та уточнити функціональність системи. Це знижує кількість помилок і необхідність переробки на наступних етапах.

Оптимізація ресурсів: чітке розуміння вимог дозволяє ефективно планувати ресурси, час та бюджет, мінімізуючи витрати на зайві функції чи виправлення. Формування чітких технічних завдань: ретельно зібрані і задокументовані вимоги стають основою для створення специфікацій, які допомагають розробникам, тестувальникам та іншим учасникам команди зрозуміти, що необхідно реалізувати.

Крім цього, це допомагає полегшенню тестування та перевірки: вимоги є основою для розробки тестів, що дозволяють переконатися, що система функціонує правильно і відповідає поставленим завданням. Погано продумані вимоги можуть призвести до значних фінансових та часових втрат. Аналіз вимог допомагає виявити та оцінити потенційні ризики на ранніх етапах, що знижує ймовірність непередбачуваних проблем.

Діаграма варіантів використання (use case diagram) потрібна для графічного відображення взаємодії користувачів із системою. Вона показує основні функції (варіанти використання), які система повинна виконувати для досягнення цілей користувачів (акторів).

Діаграма допомагає:

1. Описати функціональні вимоги –демонструє, що саме система повинна робити з точки зору користувача.
2. Ідентифікувати акторів і їх ролі – визначає, хто взаємодіє з системою, і які дії можуть виконувати.
3. Зрозуміти обсяг проекту – допомагає визначити, які функції є пріоритетними.
4. Полегшити спілкування – забезпечує всім учасникам проекту загальне бачення системи і її функціоналу.

Таким чином, діаграма варіантів використання є корисним інструментом для спільного обговорення та планування функціональності системи (рис. 3.1).

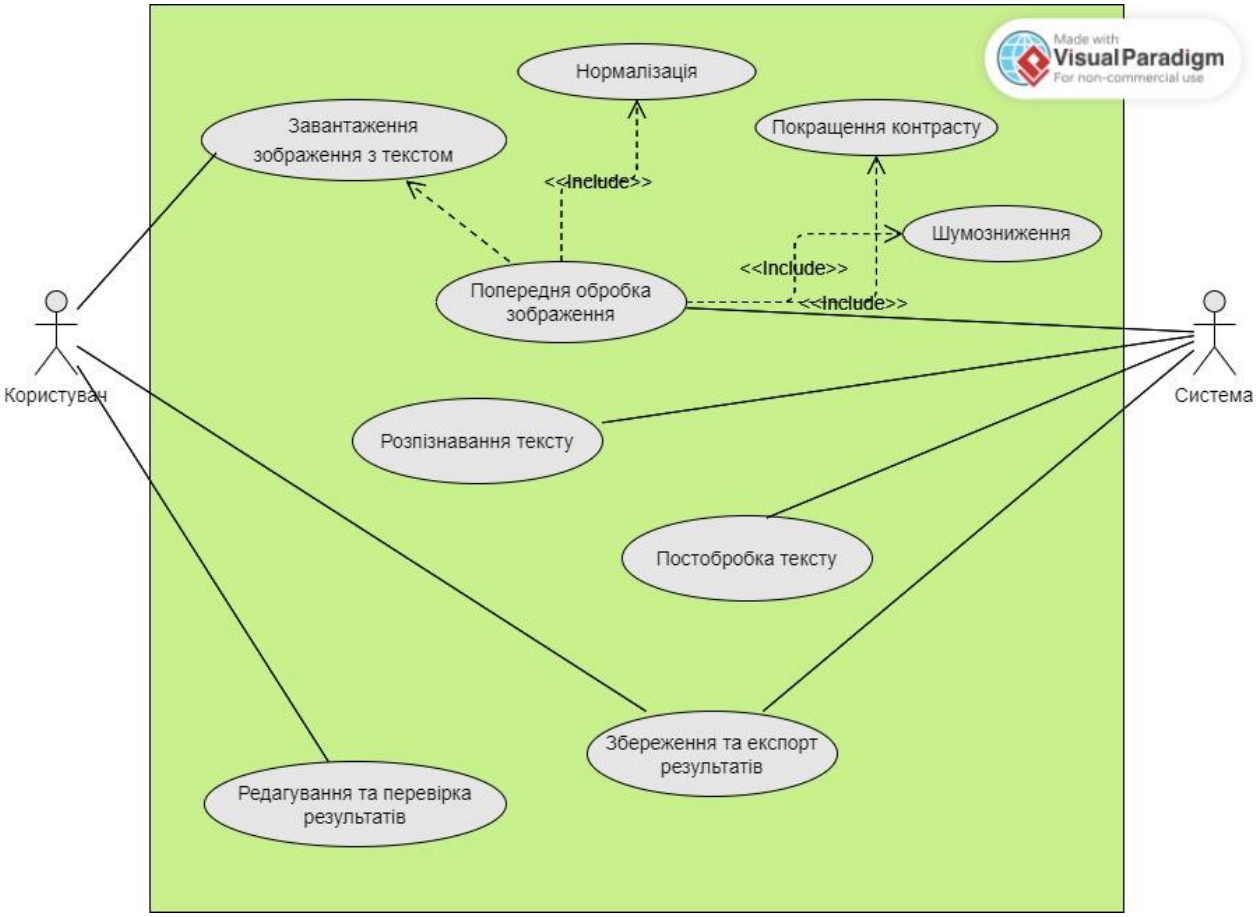


Рисунок 3.1 – Діаграма варіантів використання до системи

На діаграмі представлено 9 основних функцій, які будуть виконувати користувач і сама система. Опишемо кожен варіант використання більш детально: його назва, хто виступає основним актором, основний сценарій та альтернативний сценарій.

ВВ №1 «Завантаження зображення з текстом»

Актори: Користувач.

Основний сценарій:

1. Користувач вибирає зображення з текстом із локального пристрою або завантажує з мережі.
2. Система приймає зображення та конвертує його у формат, зручний для подальшої обробки (наприклад, чорно-білий, якщо потрібно).

Альтернативний сценарій: якщо формат зображення не підтримується, система повідомляє користувача про необхідність завантаження іншого формату (JPEG, PNG, TIFF) і пропонує повторити завантаження.

ВВ №2 «Попередня обробка зображення»

Актори: Система.

Основний сценарій:

1. Система застосовує попередню обробку до завантаженого зображення:
2. Розтягнення контрасту для покращення видимості тексту.
3. Застосування шумозниження, якщо зображення містить артефакти або дефекти.
4. Виконання порогової сегментації для відокремлення тексту від фону.
5. Після обробки система зберігає результат для подальшого розпізнавання.

Альтернативний сценарій: якщо текст на зображенні все ще залишається нерозбірливим, система може повідомити про необхідність налаштування параметрів обробки (наприклад, рівня контрасту) або пропонує користувачеві перезавантажити зображення з кращою якістю.

ВВ №3 «Розпізнавання тексту»

Актори: Система.

Основний сценарій:

1. Система виконує процес оптичного розпізнавання символів (OCR) за допомогою алгоритму, використовуючи підготовлене зображення.
2. Tesseract OCR обробляє зображення та перетворює текст на цифровий формат (рядок символів).
3. Система зберігає результат розпізнавання для подальшої обробки.

Альтернативний сценарій: якщо система не може повністю розпізнати текст, вона повертає результат із позначенням нерозпізнаних символів або частин тексту та пропонує користувачу перевірити їх вручну.

ВВ №4 «Постобробка тексту»

Актори: Система.

Основний сценарій:

1. Система виконує очищення тексту від зайвих символів або артефактів, що могли виникнути під час розпізнавання.
2. За необхідності, система може виконати базову перевірку орфографії, коригуючи розпізнані слова.
3. Результат обробки зберігається у зручному форматі для користувача (наприклад, текстовий файл або JSON).

Альтернативний сценарій: якщо певні слова або фрази були розпізнані некоректно, система надає користувачеві можливість вручну редагувати результат або пропонує варіанти виправлення.

ВВ №5 «Збереження та експорт результатів»

Актори: Користувач, Система.

Основний сценарій:

1. Після завершення розпізнавання користувач обирає, як зберегти результат (наприклад, у текстовому файлі, PDF або JSON).

2. Система генерує вибраний формат та надає можливість завантажити файл.

Альтернативний сценарій: Якщо збереження у вибраному форматі неможливе через обмеження, система пропонує альтернативні формати зберігання.

ВВ №6 «Редагування та перевірка результатів»

Актори: Користувач.

Основний сценарій:

1. Користувач переглядає розпізнаний текст, редагує або виправляє слова, які були розпізнані неточно.
2. Система надає простий інтерфейс для внесення змін, якщо це передбачено функціоналом.
3. Після редагування користувач може зберегти остаточний результат.

Альтернативний сценарій: якщо користувач помічає багато помилок, він може повернутися до попередніх кроків (наприклад, виконати повторну обробку зображення з іншими параметрами) і повторити розпізнавання.

Ці функції та сценарії дають чітке розуміння основного функціоналу системи OCR для розпізнавання тексту з зображень та ролі, яку відіграє користувач у процесі обробки, редагування й збереження результатів.

Для варіанту використання «Попередня обробка зображення» можна виділити три додаткові варіанти: нормалізація, покращення контрасту та шумозниження. Кожен з них виконує окрему задачу для поліпшення якості зображення перед основним процесом розпізнавання тексту.

ВВ №7 «Нормалізація зображення»

Актори: Користувач.

Основний сценарій:

1. Користувач завантажує зображення із текстом у систему.
2. Система аналізує діапазон значень інтенсивності пікселів зображення.

3. Зображення нормалізується, щоб забезпечити узгодженість між різними зображеннями (наприклад, вирівнюється яскравість і контраст).
4. Нормалізоване зображення зберігається або передається на наступний етап обробки.

Альтернативний сценарій: якщо зображення має серйозні спотворення, система повідомляє про можливу низьку якість і пропонує завантажити інше зображення або продовжити з поточним.

ВВ №8 «Покращення контрасту»

Актори: Користувач.

Основний сценарій:

1. Користувач завантажує зображення, на якому текст важко розпізнати через низький контраст.
2. Система аналізує контраст зображення і застосовує методи його підвищення (наприклад, розтягнення гістограми чи вирівнювання гістограми).
3. Покращене за контрастом зображення стає більш чітким, що підвищує ймовірність успішного розпізнавання тексту.
4. Зображення з підвищеним контрастом зберігається для подальшої обробки або аналізу.

Альтернативний сценарій: якщо система виявляє, що збільшення контрасту призводить до втрати інформації (наприклад, якщо зникають важливі деталі), вона повідомляє користувача і пропонує зберегти оригінальне зображення або спробувати інший метод.

ВВ №9 «Шумозниження»

Актори: Користувач.

Основний сценарій:

1. Користувач завантажує зображення, яке містить шум (наприклад, плями, артефакти, спричинені скануванням або фотозйомкою).

2. Система застосовує фільтрацію для зменшення шуму (наприклад, середнє згладжування, медіанний фільтр або гаусівське розмиття).
3. Очищене від шуму зображення передається для подальшого розпізнавання тексту.

Альтернативний сценарій: якщо зображення має високий рівень шуму, і його очищення може призвести до втрати важливої інформації, система може зберегти кілька варіантів обробленого зображення або запропонувати користувачу завантажити нове.

3.2 Загальна архітектура системи

Для проектування загальної архітектури OCR-системи на основі класичних методів можна виділити кілька основних компонентів та описати їхні зв'язки (рис. 3.2). Така система включає етапи, починаючи з отримання вхідного зображення, його обробки та підготовки, до кінцевого розпізнавання символів і повернення результату.

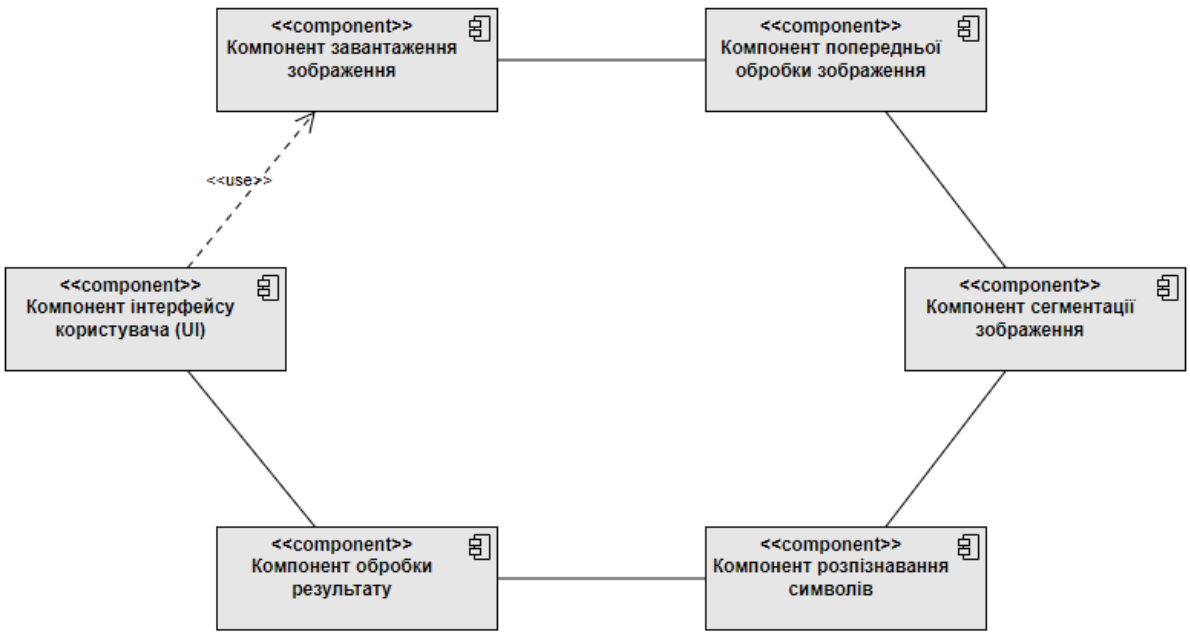


Рисунок 3.2 – Діаграма компонентів системи

Опишемо основні компоненти системи OCR:

1. Компонент завантаження зображення.

Приймає на вхід зображення від користувача (завантажене із локального комп'ютера, зроблене на камеру тощо).

Функції: перевіряє формат файлу, його розмір та забезпечує інтеграцію з іншими компонентами системи.

Зв’язки: взаємодіє з компонентом попередньої обробки зображення для передачі отриманого зображення.

2. Компонент попередньої обробки зображення.

Виконує всі необхідні операції для покращення якості зображення перед його подальшою обробкою. До його функцій входять:

- нормалізація зображення: вирівнює яскравість і контраст;
- покращення контрасту: застосовує методи розтягнення або вирівнювання гістограми;
- шумозниження: виконує фільтрацію для зменшення шуму;
- бінаризація: перетворює зображення у двоколірний формат для полегшення розпізнавання тексту.

Зв’язки: приймає вхід від компонента завантаження зображення і передає оброблене зображення компоненту сегментації.

3. Компонент сегментації зображення.

Виділяє текстові області або окремі символи для подальшого розпізнавання.

Функції:

- локалізує текстові блоки, рядки чи окремі символи на зображенні;
- передає сегментовані зони в компонент розпізнавання символів.

Зв’язки: отримує оброблене зображення від компонента попередньої обробки і передає його компоненту розпізнавання.

4. Компонент розпізнавання символів.

Опис: виконує розпізнавання символів на сегментованих областях, використовуючи методи шаблонного зіставлення або інші класичні методи.

Функції:

- порівнює кожен символ зі встановленими шаблонами для визначення відповідного символу;
- агрегує розпізнані символи у слова та рядки тексту.

Зв’язки: приймає сегментовані символи від компонента сегментації та передає розпізнаний текст компоненту обробки результату.

5. Компонент обробки результату.

Формує кінцевий текстовий результат, здійснює можливе післяоброблення. До його функцій входять:

- об’єднання розпізнаних символів в слова та речення;
- форматування тексту за необхідності;
- зберігання або експортування результату у потрібному форматі (наприклад, у .txt або .docx).

Зв’язки: отримує дані з компонента розпізнавання символів і передає кінцевий результат користувачеві.

6. Компонент інтерфейсу користувача (UI).

Забезпечує зручну взаємодію користувача із системою для завантаження зображення, перегляду проміжних етапів обробки, налаштування параметрів тощо.

Функції:

- приймає вхідні зображення від користувача;
- дозволяє налаштовувати параметри попередньої обробки, такі як рівень контрасту, спосіб бінаризації або шумозниження;
- відображає результати розпізнавання тексту.

Цей компонент інтерактивно взаємодіє з усіма компонентами системи, зокрема з компонентами завантаження зображення та обробки результату.

Архітектуру системи вирішено реалізувати за принципом мікросервісів, де кожен компонент виконує свою специфічну функцію окремо. Такий підхід дозволяє розширювати та модифікувати окремі функціональні блоки без втручання в інші. Реалізація OCR-системи за принципом мікросервісів має

свої переваги, особливо коли є потреба в гнучкості, масштабованості та можливості легко модифікувати чи оновлювати окремі компоненти системи.

3.3 Детальне проектування модулів

3.3.1 Модуль завантаження та підготовки зображень

Модуль завантаження та підготовки зображень є ключовою частиною системи, що забезпечує початкову обробку зображень для подальшого аналізу. Основне завдання модуля полягає у завантаженні зображень, конвертації їх у потрібний формат (наприклад, у відтінки сірого) та підготовці до подальшої обробки іншими модулями.

Діаграма послідовності дій (sequence diagram) для модуля завантаження та підготовки зображень (рис. 3.3) показує взаємодію між компонентами системи та послідовність виконання дій.

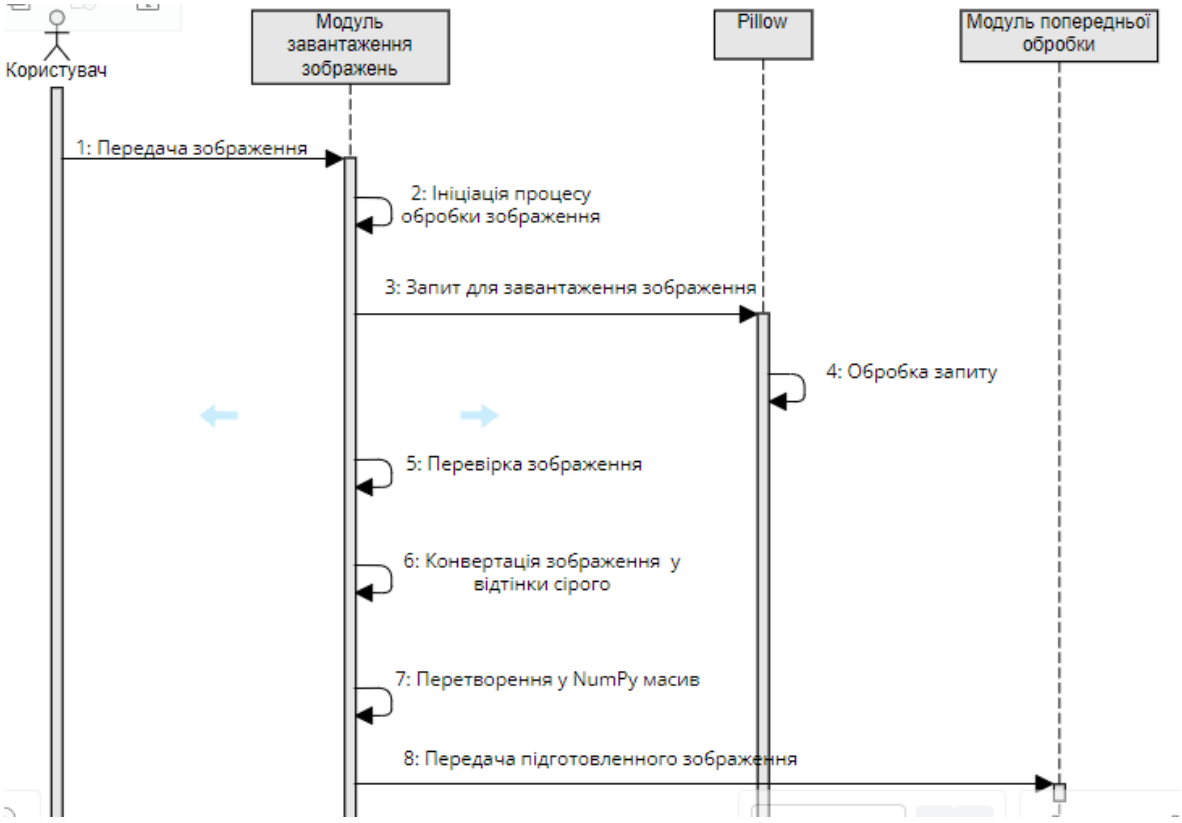


Рисунок 3.3 – Діаграма послідовності дії для роботи модуля завантаження

Компоненти діаграми:

1. Користувач – ініціатор процесу завантаження зображення.
2. Модуль завантаження зображень – компонент, що відповідає за завантаження зображення та його перетворення.
3. Pillow – бібліотека для роботи із зображеннями.
4. Модуль попередньої обробки – наступний етап, який отримує оброблене зображення для подальших дій.

Опис послідовності дій:

1. Користувач передає файл зображення в систему.
2. Модуль завантаження зображень отримує зображення і ініціює процес його обробки.

1. Модуль завантаження передає запит до Pillow для відкриття зображення (модуль використовує функцію Image.open() для завантаження зображення).
2. Бібліотека Pillow обробляє запит і відкриває зображення у відповідному форматі (jpeg, png).
3. Модуль завантаження перевіряє, чи є зображення кольоровим, і, якщо потрібно, ініціює його конвертацію у відтінки сірого.
4. Конвертація зображення у відтінки сірого — це частина логіки, яка знаходиться в модулі завантаження, якщо зображення не відповідає потрібному формату для подальшої обробки.
5. Перетворення зображення у NumPy масив — також частина цього ж модуля, оскільки саме тут зображення перетворюється у формат, зручний для обробки бібліотеками, такими як OpenCV або Scikit-Image.
6. Передача масиву до іншого модуля, наприклад, до модуля попередньої обробки, відбувається на останньому етапі завантаження та підготовки зображення.

Кроки 6 і 7 є самовикликами, які показують, що модуль виконує власну обробку, не взаємодіючи з іншими компонентами або зовнішніми сервісами.

3.3.2 Модуль обробки зображень

Модуль обробки зображень відповідає за покращення якості зображення і підготовку його до подальшого етапу, де здійснюється розпізнавання тексту. Він виконує кілька кроків попередньої обробки, що включають збільшення контрастності, фільтрацію шуму, морфологічні операції та порогову сегментацію. Діаграма активності (рис. 3.4) демонструє процес роботи модуля обробки зображення з поясненнями до виклику відповідних методів.

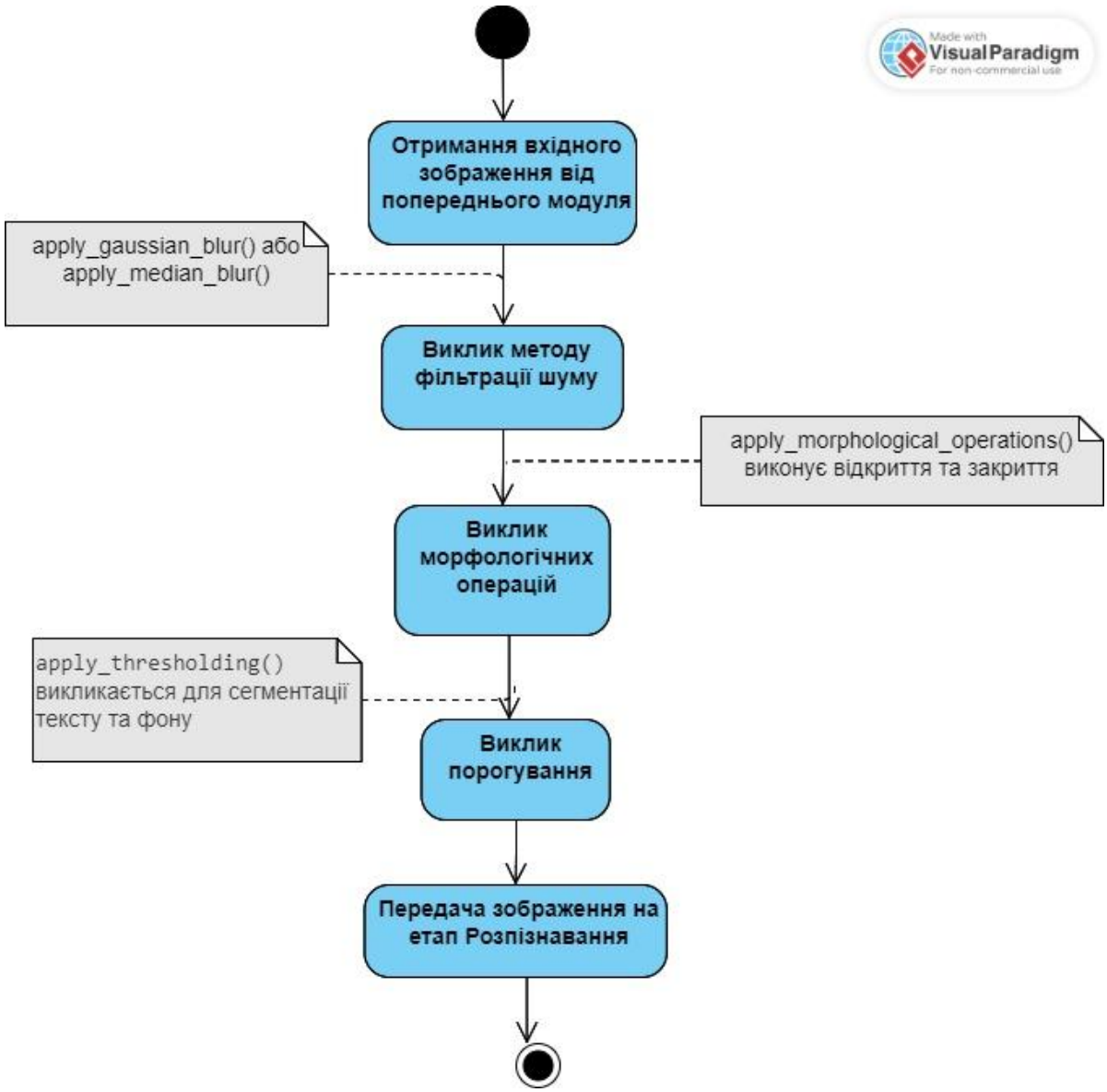


Рисунок 3.4 – Діаграма активності для модуля обробки зображення

Послідовність дій цього модуля наступна:

1. Модуль обробки зображень отримує вхідне зображення (отримане від модуля завантаження).
2. Модуль обробки зображень викликає внутрішні функції для кожного етапу обробки:
   - нормалізація зображення – підвищення контрасту;
   - шумозниження – згладжування зображення через Gaussian або Median Blur;
   - морфологічні операції – відкриття та закриття для корекції зображення;
   - порогування – перетворення зображення в бінарне.

Після кожного кроку модуль передає результати на наступний етап обробки.

3.3.3 Модуль розпізнавання тексту

Для відображення процесу роботи модуля розпізнавання тексту найкраще підходить діаграма послідовності дії, яка демонструє взаємодію між об'єктами або модулями системи у часі (рис. 3.5). Вона добре підходить, якщо потрібно продемонструвати, як налаштування параметрів та виклик функцій впливають на загальний процес.

Об’єктами (учасниками) процесу є:

- користувач, що ініціює процес розпізнавання тексту на зображенні;
- модуль завантаження зображення, який завантажує оброблене зображення для розпізнавання тексту;
- модуль налаштування параметрів, який визначає специфічні параметри для Tesseract, такі як мова розпізнавання та режим аналізу.
- Tesseract OCR безпосередньо виконує процес оптичного розпізнавання тексту;
- модуль обробки результатів обробляє текст після розпізнавання, видаляє зайві символи і надає користувачу результат.

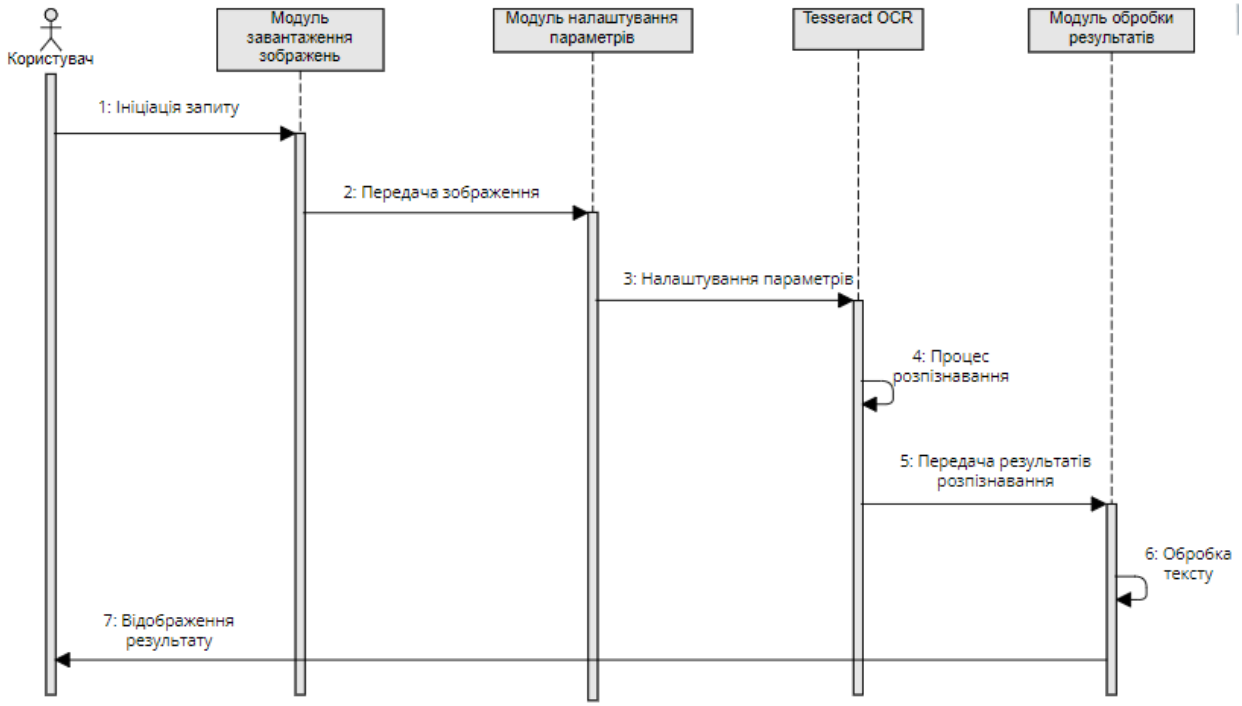


Рисунок 3.5 – Діаграма послідовності дії для модулю розпізнавання тексту

На цій діаграмі першим кроком користувач ініціює процес розпізнавання, надсилаючи запит на обробку зображення. Модуль завантаження зображення отримує цей запит і завантажує оброблене зображення. Після завантаження зображення, модуль завантаження зображення передає це зображення до модуля налаштування параметрів для налаштування специфічних опцій.

На третьому кроці модуль налаштування параметрів визначає мову, яку Tesseract повинен розпізнавати та режим розпізнавання тексту (рядки, символи, блоки тексту). Далі Tesseract OCR виконує оптичне розпізнавання тексту, аналізуючи зображення та перетворюючи його в текстові дані.

Після завершення розпізнавання, Tesseract OCR передає сирий текст до модуля обробки результатів та модуль обробки результатів виконує очищення тексту (видаляє зайві символи, пробіли, непотрібні рядки і може виконати корекцію помилок або постобробку тексту). Оброблений текст може бути переданий далі.

Ця діаграма показує, як користувач ініціює процес розпізнавання, а зображення проходить через різні етапи обробки, перш ніж Tesseract виконає OCR і надасть текст.

3.3.4 Модуль постобробки

Для демонстрації алгоритму роботи модуля постобробки обрано діаграму активності (рис 3.6).

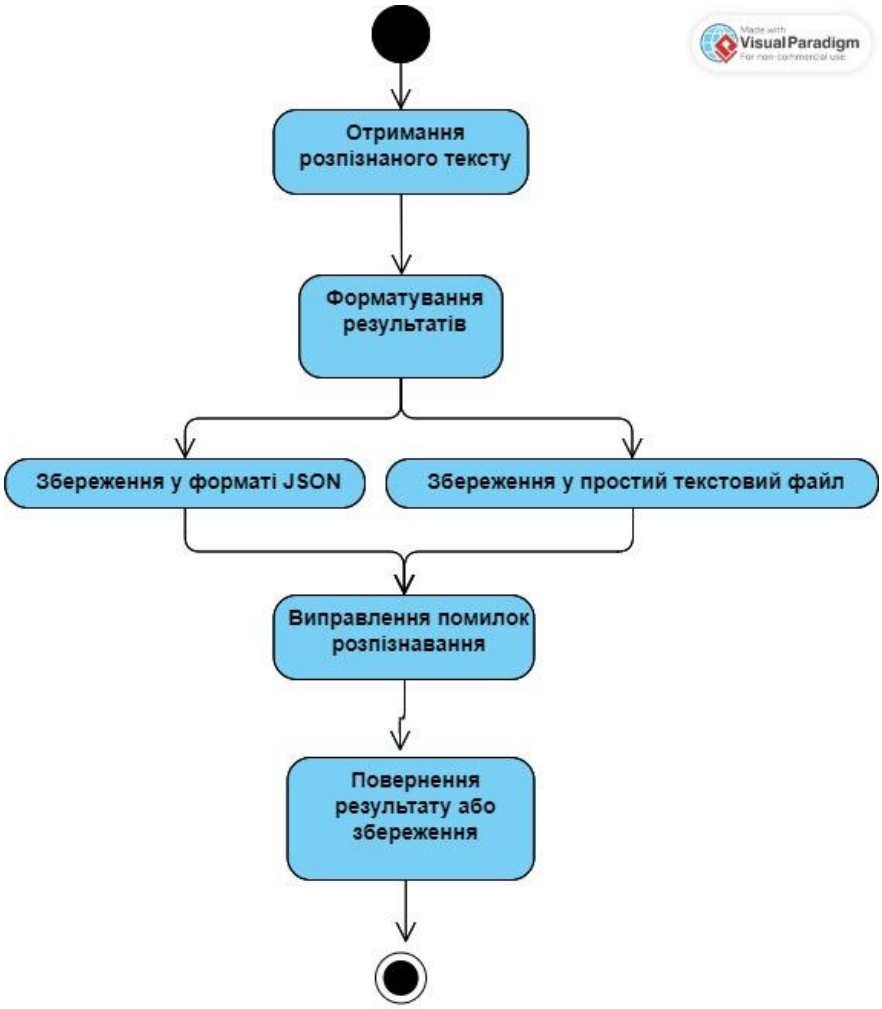


Рисунок 3.6 – Алгоритм роботи модуля постобробки

Алгоритм роботи модуля постобробки має наступні кроки:

1. Модуль постобробки отримує результат роботи Tesseract OCR – сирий текст, який може містити помилки або некоректно розпізнані символи.

2. Форматування результатів: модуль постобробки перетворює текст у відповідний формат залежно від вимог системи або запиту користувача:
   - збереження у форматі JSON (структурує текст у вигляді JSON об’єкта, що містить ключі та значення, наприклад, різні розділи тексту або окремі блоки можуть зберігатися у відповідних полях);
   - збереження у простий текстовий файл (текст зберігається у стандартний файл .txt без додаткових метаданих).
3. Модуль виконує базову перевірку та корекцію тексту:
   - орфографічна перевірка;
   - модуль виправляє поширені помилки або пропонує варіанти виправлення;
   - фільтрація зайвих символів, а саме видаляються або коригуються некоректно розпізнані символи, які можуть виникати внаслідок шуму або артефактів на зображенні (зайві пробіли, символи, що не відповідають тексту);
   - корекція форматування, тобто очищення тексту від зайвих символів або вирівнювання рядків, якщо OCR помилково злив кілька рядків у один.
4. Оброблений текст відправляється користувачу через інтерфейс та збереження у файл.

4 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

4.1 Реалізація основних функцій

Детальний опис кожної функції пояснює, як і чому використовуються певні методи та бібліотеки. Це допомагає розробникам і майбутнім користувачам системи швидко розуміти її структуру, алгоритми та логіку. Чіткий опис функцій дозволяє краще розробити тест-кейси, спрямовані на виявлення помилок або неточностей.

4.4.1 Опис функцій модуля завантаження та попередньої обробки

Почнемо з реалізації модуля завантаження та попередньої обробки зображення з коментарями, щоб було зрозуміло, як працює кожен етап. Модуль завантаження та підготовки зображень використовує бібліотеку Pillow для завантаження та базової обробки зображень, бібліотеку NumPy для роботи з масивами зображень та OpenCV для подальшої обробки зображень:

```
from PIL import Image
import numpy as np
import cv2
```

Функція завантажує зображення, конвертує його у відтінки сірого та готує для обробки. Параметр функції «image_path (str)» – шлях до зображення. Повертає вона зображення у вигляді масиву numpy для подальшої обробки:

```
def load_and_preprocess_image(image_path):
# Завантаження зображення за допомогою Pillow
    image = Image.open(image_path)

# Конвертація в відтінки сірого, якщо зображення кольорове
    image = image.convert("L") #"L" означає grayscale

# Перетворення зображення в масив numpy для подальшої
обробки з OpenCV
    image_np = np.array(image)
    return image_np
```

```
    image_path = 'your_image.png'  # Заміни на шлях до власного
зображення
    image_np = load_and_preprocess_image(image_path)
```

Під час обробки контрасту та фільтрація шуму можна додати функції для підвищення контрасту і фільтрації шуму (за допомогою нормалізації гістограми) перед передачею зображення в OCR. Функція enhance_contrast повертає оброблене зображення з підвищеним контрастом, її параметр image_np (numpy.ndarray) – це зображення у вигляді масиву numpy.

```
def enhance_contrast(image_np):
   # Розтягнення контрасту
    return cv2.equalizeHist(image_np)
```

Видалення шуму з використанням Gaussian Blur виконує функція denoise_image(). Вона повертає зображення з видаленим шумом:

```
def denoise_image(image_np):
 return cv2.GaussianBlur(image_np, (5, 5), 0)
#(5, 5) - розмір ядра

# Підготовка зображення перед OCR
image_contrast = enhance_contrast(image_np)
image_denoised = denoise_image(image_contrast)
```

Розмір ядра вказує на розмір області, над якою застосовується згладжування для кожного пікселя зображення. Ядро — це прямокутна матриця, яка переміщається по всьому зображенню, застосовуючи фільтр (у цьому випадку – Gaussian Blur) до кожної області.

(5, 5) тут означає, що ядро має розміри 5x5 пікселів. Це число визначає "радіус" впливу фільтра навколо кожного оброблюваного пікселя. Чим більше ядро, тим сильніше буде згладжування, оскільки кожен піксель усереднюватиметься з ширшою областю сусідніх пікселів, що особливо корисно для зменшення великого шуму.

Менше ядро (наприклад, 3x3) призведе до слабшого ефекту розмиття, зберігаючи більше деталей, але видаляючи меншу кількість шуму. Аргумент 0 в кінці виконує функцію параметра стандартного відхилення для обчислення розмиття. Якщо він дорівнює нулю, OpenCV самостійно обчислює його на основі розміру ядра, що в більшості випадків підходить.

Наступний крок, це інтеграція з Tesseract для OCR. Для цього використовується бібіліотека «pytesseract». Вона виконує розпізнавання тексту та повертає розпізнаний текст. Її парамметри:

- image_np (numpy.ndarray): зображення у вигляді масиву numpy;
- lang (str): мова для розпізнавання (за замовчуванням 'eng').

```
import pytesseract  #Для взаємодії з Tesseract OCR

def perform_ocr(image_np, lang='eng'):
# Виклик Tesseract OCR з параметром мови
     ocr_result = pytesseract.image_to_string(image_np,
lang=lang)
     return ocr_result # Розпізнання тексту
     recognized_text = perform_ocr(image_denoised)
     print("Розпізнаний текст:", recognized_text)
```

Всі результати зручно зберігати у форматі JSON (функція save_results_to_json):

```
import json  # Для збереження результатів у JSON

def save_results_to_json(text, output_path='result.json'):
 # Форматування даних для JSON
    data = {"recognized_text": text}

    # Запис у JSON файл
    with open(output_path,'w',encoding='utf-8') as
json_file:
    json.dump(data, json_file, ensure_ascii=False, indent=4)
    print(f"Результати збережені в {output_path}")

     # Збереження результату
     save_results_to_json(recognized_text)
```

4.4.2 Опис функцій модуля обробки зображення

Наступний модуль – модуль обробки зображення, який забезпечить покращення якості зображення для більш точного розпізнавання тексту. В цьому модулі ми реалізуємо алгоритми для підвищення контрасту, фільтрації шуму, морфологічних операцій та порогової сегментації. Це допоможе максимально відокремити текст від фону та підготувати зображення для OCR.

Підвищення контрасту зображення за допомогою розтягнення гістограми тут виконує функція enhance_contrast(). Вона отримує зображення та повертає його з покращеним контрастом:

```
import cv2
def enhance_contrast(image_np):

 # Розтягнення контрасту return
      cv2.equalizeHist(image_np)
```

Порогова сегментація для виділення тексту з фону виконує функція apply_threshold(). Використовуємо адаптивне порогування для динамічної обробки різних освітлених областей:

```
def apply_threshold(image_np):
   return cv2.adaptiveThreshold(image_np, 255,
   cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)
```

Видалення шуму з використанням Gaussian Blur – це задача denoise_image().

```
def denoise_image(image_np):
       return cv2.GaussianBlur(image_np, (5, 5), 0)
```

Застосування морфологічних операцій для покращення розділення тексту від фону:

```
def morphological_operations(image_np):
```

```
    # Ядро 3x3 для морфологічних операцій
      kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))

    # Морфологічне відкриття для видалення дрібних шумів
      image_opened = cv2.morphologyEx(image_np, cv2.MORPH_OPEN,
        kernel)

    # Морфологічне закриття для з'єднання розірваних частин
тексту
     return cv2.morphologyEx(image_opened, cv2.MORPH_CLOSE,
        kernel)
```

Повна обробка зображення для OCR: підвищення контрасту, фільтрація шуму, порогове значення та морфологічні операції – за задачі функції preprocess_for_ocr (). Вона повертає попередньо оброблене зображення для OCR.

```
def preprocess_for_ocr(image_path):
    image = load_and_preprocess_image(image_path)
    image_contrast = enhance_contrast(image)
    image_denoised = denoise_image(image_contrast)
    image_thresholded = apply_threshold(image_denoised)
    image_processed =
          morphological_operations(image_thresholded)

    return image_processed
```

Отже, використовуємо розтягнення гістограми (equalizeHist) для рівномірного розподілу яскравості пікселів і виділення тексту на фоні. «GaussianBlur» допомагає згладити зображення та видалити дрібні шуми. Адаптивне порогування (adaptiveThreshold) дозволяє динамічно виділяти текст у різних умовах освітлення. Морфологічні операції: morphologyEx з операціями відкриття та закриття дозволяє покращити чіткість тексту, видаляючи шуми та об'єднуючи розірвані частини символів.

Виклик модуля для перевірки:

```
# Завантаження та обробка зображення для OCR
   image_path = 'your_image.png'
   processed_image = preprocess_for_ocr(image_path)
```

```
# Перевірка результату обробки (опційно)
   cv2.imshow("Processed Image", processed_image)
   cv2.waitKey(0)
   cv2.destroyAllWindows()
```

Модуль обробки завершує підготовку зображення, яке потім можна передати в OCR для розпізнавання тексту.

4.4.3 Опис функцій модуля розпізнавання тексту

В цьому модулі вже оброблене зображення надсилається в Tesseract OCR для витягу тексту. Модуль також включатиме налаштування параметрів Tesseract для підвищення точності розпізнавання. Отже, розпізнавання тексту з обробленого зображення за допомогою Tesseract OCR виконує функція recognize_text().

Парамметри фннкції:

- processed_image (numpy.ndarray): попередньо оброблене зображення;
- lang (str): мова для розпізнавання (наприклад, 'eng' для англійської);
- psm (int): режим сегментації сторінки для Tesseract (за замовчуванням 6).

```
import pytesseract
def recognize_text(processed_image, lang='eng', psm=6):

# Налаштування параметрів Tesseract для покращення
розпізнавання
    config = f'--oem 3 --psm {psm}'

# Розпізнавання тексту з використанням Tesseract
    recognized_text =
      pytesseract.image_to_string(processed_image,
      lang=lang, config=config)

    return recognized_text

# Приклад виклику функції
image_path = 'processed_image.png'  # шлях до обробленого
зображення

# Попередня обробка зображення
processed_image = preprocess_for_ocr(image_path)
```

```
text = recognize_text(processed_image)
print("Розпізнаний текст:", text)
```

Розпізнавання тексту з Tesseract: pytesseract.image_to_string обробляє вхідне зображення, використовуючи Tesseract OCR, і повертає розпізнаний текст у вигляді рядка. Тут до налаштування параметрів входять:

- режим сегментації сторінки (psm, Page Segmentation Mode) – дозволяє Tesseract розпізнавати текст відповідно до різних структур, від окремих слів до повних сторінок тексту (наприклад, psm=6 підходить для обробки блоків тексту);
- мова розпізнавання (lang) – підтримка конкретної мови або комбінації мов для більш точного розпізнавання. Це налаштування є корисним, якщо система працює з різними мовами.

Отриманий текст можна відобразити або зберегти для подальшої обробки. Цей модуль завершує основний процес розпізнавання тексту.

4.4.4 Опис функцій модуля постобробки

Цей модуль відповідає за форматування результатів розпізнавання та виконання базових виправлень орфографії. Функція postprocess_text() займається постобробкою розпізнаного тексту: форматування та корекція орфографії. Вона працює з текстом, отриманий після розпізнавання:

```
import json
import re
from spellchecker import SpellChecker

def postprocess_text(recognized_text, output_format='text'):
```

Спочатку потрібні ініціалізація spellchecker для корекції орфографії та розбиття тексту на слова для перевірки орфографії:

```
  spell = SpellChecker()
   words = recognized_text.split()
```

```
        corrected_words = []

        for word in words:
            # Якщо слово неправильно написане, виправити його
            if word not in spell:
                corrected_word = spell.correction(word)
                corrected_words.append(corrected_word)
            else:
                corrected_words.append(word)
```

З'єднання слів у скоригований текст та збереження результатів у потрібному форматі:

```
        corrected_text = ' '.join(corrected_words)
        if output_format == 'json':
            output = json.dumps({"recognized_text":
corrected_text}, ensure_ascii=False, indent=4)
        else:
            output = corrected_text

        return output
```

Приклад використання:

```
recognized_text = "Ths is sme txt wth erors." #розпізнаний
                        текст з помилками
formatted_text = postprocess_text(recognized_text,
                       output_format='json')
print("Відформатований та скоригований текст:",
                       formatted_text)
```

Тобто, після ініціалізації SpellChecker з бібліотеки «pyspellchecker», щоб перевіряти розпізнаний текст на орфографічні помилки розбиваємо текст на окремі слова, і для кожного слова перевіряється правильність написання. Якщо слово виявлено як помилкове, застосовується метод correction(), щоб знайти найімовірнішу заміну. Модуль завершує обробку результату та повертає його у вибраному форматі. За потреби, результат можна зберегти у файл або вивести на екран для користувача.

4.2 Функціональне тестування системи

Для тестування кожного модуля OCR-системи розроблено план функціонального тестування, яке перевіряє чи кожен модуль працює згідно з вимогами та коректно обробляє дані на кожному етапі.

План функціонального тестування OCR-системи для модулю завантаження та підготовки зображень містить 3 тест-кейси. Мета тестування: перевірити, чи модуль коректно завантажує зображення, змінює його розмір, якщо потрібно, і правильно перетворює зображення у відтінки сірого. Нижче представлено опис кожного з них [14].

TC1 «Завантаження кольорового зображення та конвертація у відтінки сірого».

Передумови: є кольорове зображення, доступне для завантаження.

Основні кроки:

1. Завантажити зображення за допомогою модуля завантаження.
2. Виконати конвертацію у відтінки сірого.

Очікуваний результат: зображення успішно завантажено та конвертовано у відтінки сірого, без втрати якості.

TC2 «Завантаження пошкодженого або некоректного зображення».

Передумови: є пошкоджене або некоректне зображення.

Основні кроки: завантажити зображення за допомогою модуля завантаження.

Очікуваний результат: модуль видає відповідне повідомлення про помилку завантаження.

TC3 «Завантаження зображення у відтінках сірого».

Передумови: наявність зображення у відтінках сірого.

Основні кроки: завантажити зображення та перевірити його формат.

Очікуваний результат: зображення завантажується без додаткової конвертації.

Для модулю обробки зображень також складено 3 тест-кейси. Мета тестування: перевірити, чи модуль застосовує контрастність, шумозниження, морфологічні операції та порогове значення для поліпшення якості зображення.

ТС4 «Застосування розтягнення контрасту та нормалізації».

Передумови: є зображення з низьким контрастом.

Основні кроки:

1. Завантажити зображення у модулі обробки.
2. Виконати операцію розтягнення контрасту та нормалізацію.

Очікуваний результат: контраст зображення покращено, текст чіткіше відокремлений від фону.

ТС5 «Видалення шуму зображення за допомогою Gaussian Blur».

Передумови: є зображення з шумом.

Основні кроки:

1. Завантажити зображення у модулі обробки.
2. Виконати фільтрацію з Gaussian Blur.

Очікуваний результат: шум на зображенні зменшений, текст залишається чітким.

ТС6 «Порогова сегментація тексту».

Передумови: є зображення з низьким контрастом між текстом і фоном, що ускладнює розпізнавання.

Основні кроки:

1. Завантажити зображення у модуль обробки.
2. Застосувати порогову сегментацію (бінаризацію), щоб виділити текст.

Очікуваний результат: зображення перетворене на двоколірний формат, у якому текст має чіткий контраст із фоном.

Наступні три тест-кейси створені для модулю розпізнавання тексту. Мета тестування: перевірити, чи модуль розпізнає текст і налаштовує параметри для покращення точності розпізнавання.

ТС7 «Виконання OCR на зображенні з чітким текстом».

Передумови: є зображення з текстом високої якості, де символи добре відокремлені та легко читаються.

Основні кроки:

1. Завантажити зображення у модуль розпізнавання.
2. Виконати OCR за допомогою Tesseract.

Очікуваний результат: усі символи на зображенні розпізнані правильно, текст повертається без помилок.

ТС8 «OCR на зображенні з текстом низької якості»

Передумови: є зображення з текстом низької якості (наприклад, розмитий текст або низький контраст).

Основні кроки:

1. Завантажити зображення у модуль розпізнавання.
2. Виконати OCR за допомогою Tesseract.

Очікуваний результат: текст розпізнаний частково. Якщо OCR не може розпізнати частину тексту, він повертає текст з можливими пропусками або помилками, але система не зупиняється аварійно.

ТС9 «Зміна мови розпізнавання на нерідну для зображення мову».

Передумови: є зображення з текстом однією мовою (наприклад, англійською), а для розпізнавання обрана інша мова (наприклад, іспанська).

Основні кроки:

1. Завантажити зображення у модуль розпізнавання.
2. Вибрати мову розпізнавання, яка не відповідає мові тексту на зображенні.
3. Виконати OCR за допомогою Tesseract з налаштованою мовою.

Очікуваний результат: система попереджає про низьку точність розпізнавання або надає текст із помилками, оскільки обрана мова не відповідає фактичній мові тексту.

Останні тест-кейси для модулю постобробки. Мета тестування: перевірити, чи модуль коректно форматуватиме результати і виправлятиме орфографічні помилки.

ТС10 «Збереження результатів у JSON-форматі».

Передумови: розпізнаний текст готовий до збереження, система має підтримку формату JSON.

Основні кроки:

1. Завантажити розпізнаний текст у модуль постобробки.
2. Виконати збереження тексту у форматі JSON.

Очікуваний результат: дані зберігаються у форматі JSON зі структурою, що зручно відображає текстові блоки та метадані (наприклад, час розпізнавання або обрану мову), і готові до передачі чи зберігання у файлі.

ТС11 «Форматування результатів у вигляді простого тексту».

Передумови: Розпізнаний текст готовий до виведення у простому текстовому форматі.

Основні кроки:

1. Завантажити розпізнаний текст у модуль постобробки.
2. Виконати форматування тексту у вигляді звичайного текстового рядка без додаткової структури.

Очікуваний результат: вихідний текст зберігається у простому текстовому форматі, готовий до виведення на екран або збереження у текстовому файлі (.txt).

ТС12 «Виправлення орфографічних помилок у розпізнаному тексті».

Передумови: розпізнаний текст містить потенційні орфографічні помилки, доступний інструмент для перевірки орфографії.

Основні кроки:

1. Завантажити розпізнаний текст у модуль постобробки.
2. Виконати перевірку тексту на наявність орфографічних помилок.
3. Автоматично виправити знайдені помилки.

Очікуваний результат: основні орфографічні помилки виправлені, і текст приведений до правильної форми, готовий до подальшого використання.

Всього 12 тест-кейсів, які охоплюють основні функціональні можливості системи OCR (відповідно до описаної раніше діаграми варіантів використання). Ці тест-кейси забезпечують перевірку всіх ключових етапів обробки зображення, розпізнавання тексту, збереження та постобробки результатів.

Для підтвердження коректної роботи системи підготовлено декілька зображень з різним рівнем якості тексту. Якщо система успішно розпізнає тексти на тестових зображеннях, можна зроюити висновки щодо досягнення поставленної мети роботи.

Результати тестування представлені у таблицы 4.1

Таблиця 4.1 – Результати тестування

№ТС	Назва тест-кейсу	Підтвердження очікуваного результату
1	Завантаження кольорового зображення та конвертація у відтінки сірого	Підтверджено
2	Завантаження пошкодженого або некоректного зображення	Підтверджено
3	Завантаження зображення у відтінках сірого	Підтверджено
4	Застосування розтягнення контрасту та нормалізації	Підтверджено
5	Видалення шуму зображення за допомогою Gaussian Blur	Підтверджено
6	Порогова сегментація тексту	Підтверджено
7	Виконання OCR на зображенні з чітким текстом	Підтверджено

Продовження таблиці 4.1

8	OCR на зображенні з текстом низької якості	Підтверджено
9	Зміна мови розпізнавання на нерідну для зображення мову	Підтверджено
10	Збереження результатів у JSON-форматі	Підтверджено
11	Форматування результатів у вигляді простого тексту	Підтверджено
12	Виправлення орфографічних помилок у розпізнаному тексті	Підтверджено

4.3 Експериментальне тестування роботи системи

Експериментальне тестування системи – це метод тестування, що полягає у проведенні досліджень і перевірок, щоб оцінити, як система поводиться в різних ситуаціях. На відміну від класичних методів тестування з чітко визначеними сценаріями та очікуваними результатами, експериментальне тестування є більш гнучким і орієнтованим на дослідження нових або нетипових аспектів роботи системи.

На першому кроці роботи з системою обираємо потрібний файл для аналізу «test_img.jpeg» (рис. 4.1).

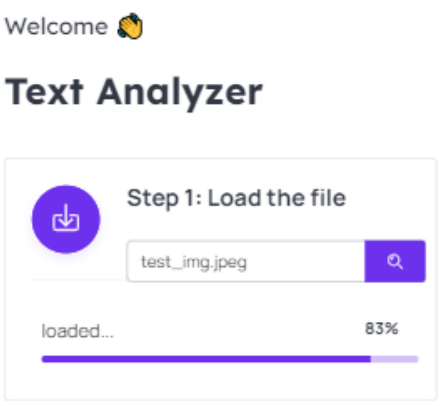


Рисунок 4.1 – Форма для завантаження даних

Після завантаження файлу система запускає процес обробки зображення та його аналізу відповідно до описаних вище алгоритмів. Для користувача висвічується назва етапу роботи системи на показується рівень його виконання в реальному часі (рис. 4.2).

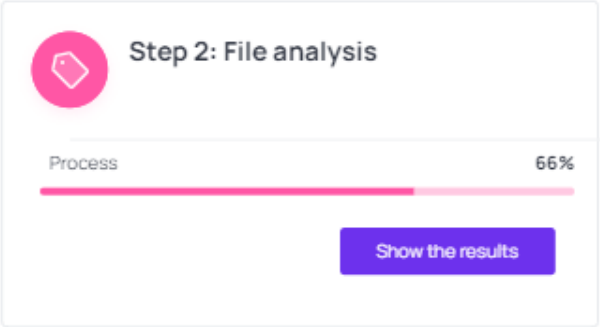


Рисунок 4.2 – Скріншот другого етапу роботи системи

Після завершення аналізу система відображає повідомлення щодо успішності перевірки, розмір завантаженого файлу та результати розпізнавання тексту (рис. 4.3). Вхідний файл представлено на рис. 4.4.

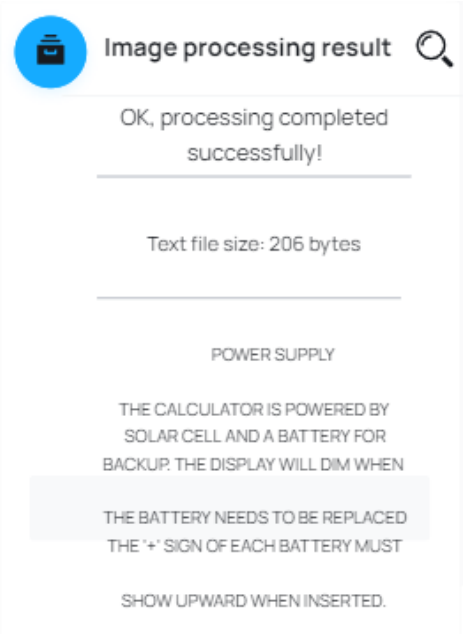


Рисунок 4.3 – Скріншот форми з результатами аналізу

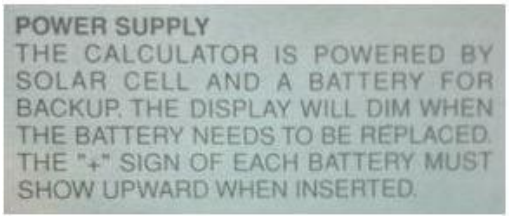
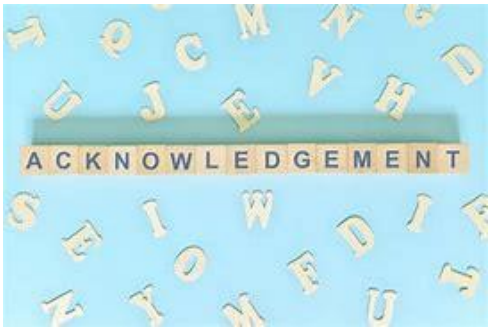


Рисунок 4.4 – Вхідний файл «test_img.jpeg»

OCR-систему протестовано на зображеннях з різними роздільними здатностями, кутами нахилу тексту, фоном, контрастом та різноманітними шрифтами. Для цього використаноо два тестових зображення, які по черзі завантажимо у систему для аналізу.

Для перевірки коректної роботи проаналізуємо ще один файл «test_img_colour.jpeg», який представлено у кольорі на різним нахилом тексту (рис. 4.5а). Результат роботи системи після аналізу цього файле на рис. 4.5б.



а) б)

Рисунок 4.5 – Результати аналізу файлу «test_img_colour.jpeg»

ВИСНОВКИ

Створено систему для оптичного розпізнавання тексту на зображеннях з використанням класичних методів OCR. Для розробки системи були оглянуті методи оптичного розпізнавання символів та наведено обґрунтування вибору класичних методів обробки зображень без використання нейронних мереж та вибір технологій та інструментів розробки.

Етап проектування OCR-системи включило в себе аналіз вимог до системи за допомогою моделювання діаграми Use-Case, вибір мікросервісной архітектури та проектування його її модулів, а також алгоритми роботи кожного з них. Кожен компонент системи чітко виконує окрему функцію – від попередньої обробки до розпізнавання та збереження результатів. Це забезпечує можливість подальшого розширення функціональності та спрощує тестування й обслуговування.

До опису практичної реалізації системи додано тестування у вигляді складання 12 тест-кейсів для функціонального тестування та наведення результатів при експериментальному тестуванні з декількома файлами.

Отриманні результати дозволяють зробити наступні висновки щодо функціональності системи:

1. Система здатна ефективно розпізнавати текст на зображеннях, обробляти його, зберігати результати та виконувати базову постобробку. Це свідчить про функціональність системи для практичного застосування.
2. Завдяки модулям попередньої обробки, система може адаптуватися до різних типів зображень (різний контраст, яскравість, шум), що розширює її сферу застосування.

Висновки щодо ефективності використання класичних методів:

1. Для основних задач розпізнавання тексту на зображеннях система успішно працює без нейронних мереж. Це доводить, що класичні методи обробки зображень можуть ефективно вирішувати OCR-

завдання з прийнятним рівнем точності, особливо в умовах, де простота й продуктивність важливіші за надвисоку точність.

2. Хоча система працює добре, класичні методи можуть бути обмежені у випадках з низькоякісними або сильно зашумленими зображеннями. Нейронні мережі могли б краще впоратися з такими складними випадками, але це також потребувало б значно більше ресурсів.

Система досягає високої точності розпізнавання на чітких зображеннях, але може потребувати додаткового налаштування для складних зображень (низька якість, багато шуму). Ефективність підвищується завдяки налаштуванням Tesseract OCR, таким як обмеження мови та режиму розпізнавання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Comprehensive Guide on Optical Character Recognition (+6 Best OCR Software in 2024) . URL: https://www.docsumo.com/blogs/ocr/what-is (дата звернення 12.05.2024)
2. OCR (оптичне розпізнавання символів) – визначення, переваги, проблеми та випадки використання. URL: https://uk.shaip.com/blog/ocr-definition-benefits-challenges-and-use-cases-infographic (дата звернення 18.05.2024)
3. Вступ до оптичного розпізнавання символів (OCR) – HashDork. URL: https://hashdork.com/uk/вступ-до-OCR (дата звернення 29.05.2024)
4. Image Processing in Python: Algorithms, Tools, and Methods You Should Know. URL: https://neptune.ai/blog/image-processing-python (дата звернення 10.08.2024)
5. Що таке оптичне розпізнавання символів (OCR) і як воно працює? URL: https://uk.macgence.com/blog/what-is-optical-character-recognition-ocr-and-how-does-it-work/ (дата звернення 10.08.2024)
6. Розкриття можливостей OCR: важливість, технологія, типи, переваги та застосування. URL: https://uk.shaip.com/blog/ocr-overview-and-applications/ (дата звернення 03.09.2024)
7. Pattern Searching. URL: https://www.geeksforgeeks.org/pattern-searching/ (дата звернення 11.10.2024)
8. Advances in image processing using machine learning techniques. URL: https://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/sil2.12146 (дата звернення 11.10.2024)
9. Pillow · PyPI. URL: https://pypi.org/project/pillow/ (дата звернення 19.10.2024)

10. GitHub – tesseract-ocr/tesseract: Tesseract Open Source OCR Engine (main repository). URL: https://github.com/tesseract-ocr/tesseract (дата звернення 23.10.2024)
11. Tesseract User Manual and tessdoc. URL: tesseract-ocr.github.io (дата звернення 23.10.2024)
12. Scikit-Image: Image processing in Python. URL: https://pypi.org/project/scikit-image/ (дата звернення 23.10.2024)
13. Get Started OpenCV. URL: https://opencv.org/get-started/ (дата звернення 23.10.2024)
14. Modern Software Testing Techniques: A Practical Guide for Developers and Testers 1st ed. Edition. Istvan Forgacs, Attila Kovacs: Apress, 2024.