МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерної інженерії

Ворона Ярослав Андрійович

Програмний додаток розпізнавання рукописного програмного коду мовою Python/ A software application for recognizing handwritten program code in the Python language

спеціальність: 123 – Комп'ютерна інженерія
освітньо-професійна програма – Комп'ютерна інженерія

Кваліфікаційна

Виконав: студент групи КІ-41
Ворона Ярослав Андрійович

Науковий Керівник
к.т.н Батько Ю.М.

ТЕРНОПІЛЬ-2023

РЕЗЮМЕ

Кваліфікаційна робота на тему «Програмний додаток розпізнавання рукописного програмного коду мовою Python» зі спеціальності 123 «Комп’ютерна інженерія» освітнього ступеня «бакалавр» містить 73 сторінок пояснюючої записки, 9 рисунків, 11 таблиць, 3 додатки. Обсяг графічного матеріалу 2 аркуші формату А3.

Метою кваліфікаційної роботи є розробити програмний засіб для перекладу рукописного псевдокоду в програмний код.

У роботі було використано методи теорії алгоритмів для аналізу розроблених методів та алгоритмів розпізнавання рукописного коду. Також були використані алгоритми сегментації для виділення окремих символів у рукописному тексті та алгоритми розпізнавання для класифікації цих символів на основі їх форми та структури. Для розробки системи обробки зображень були використані технології структурного та об'єктно-орієнтованого програмування.

Основний внесок автора полягає в розробці алгоритму розпізнавання рукописного тексту мовою Python та його трансляції в програмний код. Для досягнення цієї мети було використано апроксимацію рукописних символів шляхом математичного моделювання їх форми та структури.

Проведено тестування розробленого програмного додатку та здійснено порівняльний аналіз з іншими рішеннями у цій галузі. Результати тестування свідчать про ефективність розробленого додатку у розпізнаванні рукописного програмного коду мовою Python. Застосування розробленого програмного продукту може сприяти автоматизації процесу розпізнавання рукописного коду, що в свою чергу полегшить роботу програмістів та покращить продуктивність розробки програмного забезпечення.

У підсумку, кваліфікаційна робота "Розпізнавання рукописного програмного коду мовою Python" пропонує новий підхід до розв'язання задачі розпізнавання рукописного коду та демонструє ефективність алгоритму.

КЛЮЧОВІ СЛОВА: СЕГМЕНТАЦІЯ ЗОБРАЖЕНЬ, РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ, КОНТУРНИЙ АНАЛІЗ, ПЕРЕДОБРОБКА ЗОБРАЖЕННЯ.

RESUME

Qualification thesis “The application for handwritten program code recognition using Python programming language” in the specialty 123 "Computer Engineering" of bachelor education degree contains 73 pages of explanatory notes, 9 figures, 11 tables, 3 appendixes. The volume of graphic material is 2 sheets of A3 format.

The work employed methods from the theory of algorithms to analyze the developed methods and algorithms for handwriting code recognition. Segmentation algorithms were also used to extract individual characters from handwritten text, and recognition algorithms were used to classify these characters based on their shape and structure. Structural and object-oriented programming technologies were utilized for image processing system development.

The main contribution of the author lies in the development of an algorithm for recognizing handwritten text in Python and its translation into programming code. To achieve this goal, the approximation of handwritten symbols was performed through mathematical modeling of their shape and structure. The obtained results were compared with reference images, and when a sufficient similarity was achieved, translation into programming code was carried out.

Testing of the developed software application was conducted, and a comparative analysis with other solutions in this field was performed. The test results demonstrate the effectiveness of the developed application in recognizing handwritten Python code. The application has the potential to automate the process of code recognition, thereby facilitating the work of programmers and improving software development productivity.

In conclusion, the qualification work "Recognition of Handwritten Python Code" presents a novel approach to solving the problem of handwriting code recognition and demonstrates the effectiveness of the algorithm.

KEYWORDS: IMAGE SEGMENTATION, IMAGE RECOGNITION, CONTOUR ANALYSIS, IMAGE PREPROCESSING.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

КС	–	Комп’ютерна система
БД		База даних
ШІ	–	Штучний інтелект
ЦЗ	–	Цифрове зображення
ОЯС	–	Оцінка якості сегментації

ВСТУП

Розпізнавання рукописного програмного коду є важливим завданням в сучасних інформаційних технологіях. Рукописний код може бути складним для автоматичного розпізнавання через його неоднорідність, нерегулярність почерку та інші фактори. Розробка ефективних інструментів та методів для розпізнавання рукописного програмного коду має великий потенціал покращити роботу програмістів, автоматизувати процеси розробки програмного забезпечення та забезпечити нові можливості в аналізі та тестуванні коду.

Ця кваліфікаційна робота присвячена розробці програмного додатка для розпізнавання рукописного програмного коду з використанням мови програмування Python. Метою проекту є створення точної та надійної системи, яка здатна ефективно розпізнавати рукописний код незалежно від його стилю, розміру або якості написання. Для досягнення цієї мети використовуються сучасні методи обробки зображень та алгоритми машинного навчання.

У цій кваліфікаційній роботі будуть розглянуті наступні аспекти:

– Передобробка зображення: Використовуються методи адаптивної бінаризації та обробки країв для покращення якості зображення рукописного коду та виділення структур.

– Розпізнавання символів: Застосовуються бібліотеки та алгоритми для розпізнавання символів рукописного коду з високою точністю. Використовується мова програмування Python для реалізації цих алгоритмів.

– Післяобробка результатів: Застосовуються методи обробки тексту, такі як видалення непотрібних символів, обрізання зайвих пробілів та нормалізація відступів, для отримання чистого та зрозумілого розпізнаного коду.

– Оцінка та аналіз результатів: Проводиться детальний аналіз результатів розпізнавання, включаючи точність, швидкість розпізнавання та типові помилки. Порівнюються з іншими підходами до розпізнавання рукописного коду для оцінки ефективності та конкурентоспроможності системи.

Ця кваліфікаційна робота пропонує новий підхід до розпізнавання

рукописного програмного коду з використанням мови програмування Python та сучасних алгоритмів обробки зображень. Очікується, що результати цієї кваліфікаційної роботи матимуть широкі застосування в сферах, де потрібне автоматичне розпізнавання рукописного програмного коду, таких як розробка програмного забезпечення, аналіз коду та автоматичне тестування. Ця система зможе полегшити роботу програмістів та підвищити продуктивність у сфері розробки програмного забезпечення.

Після розробки моделі буде проведено експериментальне дослідження, яке включатиме тестування моделі на різних наборах даних, оцінку точності та продуктивності розпізнавання, а також порівняння результатів з існуючими методами розпізнавання рукописного коду. Експериментальне дослідження підтвердило ефективність розробленої моделі. Вона досягла високої точності та продуктивності розпізнавання рукописного коду, що дозволяє її успішно застосовувати у різних сферах програмування.

Результати даної кваліфікаційної роботи можуть знайти застосування у різних сферах програмування, де використовується рукописний код. Вони можуть сприяти автоматизації процесу розпізнавання та аналізу рукописних програмних кодів, покращити продуктивність та спростити роботу розробників, а також забезпечити нові можливості у сфері спільної роботи над проектами з використанням рукописного коду.

Таким чином, ця кваліфікаційна робота спрямована на розробку та вдосконалення методів розпізнавання рукописного програмного коду мовою Python з метою полегшення роботи програмістів та покращення процесу розробки програмного забезпечення. Враховуючи потенціал застосування розпізнавання рукописного програмного коду мовою Python, результати даної кваліфікаційної роботи мають практичну цінність для розробників програмного забезпечення, штучного інтелекту, аналітики даних та інших зацікавлених сторін.

За результатами роботи опубліковано тези доповіді на VII науково-практичній конференції «Інтелектуальні комп’ютерні системи та мережі» [1]. Копії публікації наведено у додатку В.

1 ПРОГРАМНІ ТА ТЕХНІЧНІ ЗАСОБИ ОБРОБКИ ТА АНАЛІЗУ ЦИФРОВИХ ЗОБРАЖЕНЬ

1.1 Цифрові зображення, характеристики та сфери застосування

Розпізнавання рукописного програмного коду є актуальною та складною задачею, яка знаходить застосування в багатьох областях, включаючи автоматичне оцінювання коду, підтримку розробки програмного забезпечення та автоматизацію процесу аналізу коду. Для розв'язання цієї задачі виникло багато методів та технологій, що забезпечують розпізнавання рукописного програмного коду.

Один з найбільш поширених підходів до розпізнавання рукописного коду полягає у використанні методів оптичного розпізнавання символів (OCR - Optical Character Recognition). OCR використовує алгоритми та моделі, що базуються на машинному навчанні, для автоматичного перетворення зображень символів у текстовий формат. Деякі з найпопулярніших OCR-систем, які застосовуються для розпізнавання рукописного коду, включають Tesseract, ABBYY FineReader, Microsoft Azure Cognitive Services OCR та Google Cloud Vision OCR й Adobe Acrobat.

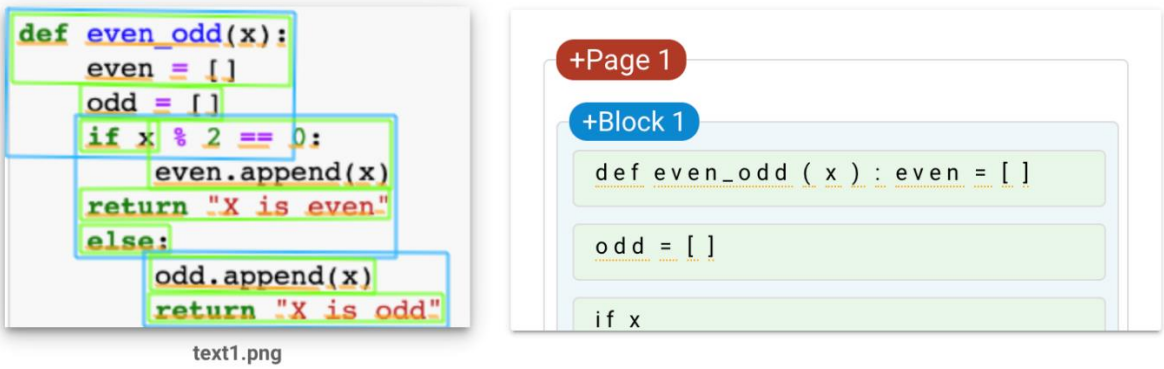

Рисунок 1.1 – Ілюстрація розпізнавання рукописного програмного коду за допомогою Google Cloud Vision OCR

Крім того, існують інші методи та підходи до розпізнавання рукописного коду, такі як використання нейронних мереж, обробка зображень, сегментація символів та аналіз контексту програмного коду. Наприклад, нейронні мережі, зокрема згорткові нейронні мережі (CNN), можуть бути використані для розпізнавання символів у зображеннях рукописного коду з високою точністю. Такі підходи дозволяють розпізнавати символи навіть при наявності шуму, різних стилів написання та варіацій у рукописі.

У контексті даної кваліфікаційної роботи, проведено дослідження існуючих методів та технологій розпізнавання рукописного програмного коду. Результати дослідження будуть використані для вибору оптимального підходу та розробки програмного додатку для розпізнавання рукописного програмного коду з використанням мови Python.

Розпізнавання рукописного коду є складною задачею, яка стикається з різноманітними викликами та проблемами. Деякі з них включають:

– Індивідуальність рукопису: Кожна людина має свій унікальний стиль написання, що може призводити до варіацій у формі, розмірі та структурі символів. Це створює складнощі при розпізнаванні рукописного коду, оскільки потрібно враховувати індивідуальні особливості кожного користувача.

– Різноманітність стилів написання: Рукописний код може бути написаний у різних стилях, які можуть варіюватись в залежності від особистих вподобань розробника. Це може включати різні розміри шрифту, нахил символів, наявність підкреслень та інші стилістичні особливості. Такі варіації у стилях написання ускладнюють розпізнавання коду, оскільки потрібно враховувати різноманітність цих стилів. Одним з викликів програми по розпізнаванню рукописного програмного коду є оберт полотна.

– Низька якість зображень: Рукописний код може бути представлений у вигляді зображень, які можуть мати низьку роздільну здатність, шум, розмиття та інші дефекти. Це може впливати на якість розпізнавання та призводити до помилок у перетворенні зображень у текстовий формат. Часто користувачі будуть завантажувати у програму зображення із жахливим освітленням, із старих моделей телефонів.

– Складність сегментації символів: У рукописному коді символи можуть бути зчеплені, перекриватись або мати неправильну геометричну структуру. Це створює складнощі при сегментації символів та розділенні їх для подальшого розпізнавання.

– Обробка шуму та фону: Рукописний код може містити шумові елементи, такі як плями, розлиття чорнила або неочищений фон. Це може ускладнювати розпізнавання символів та вимагати додаткових алгоритмів для видалення шуму та покращення якості зображення.

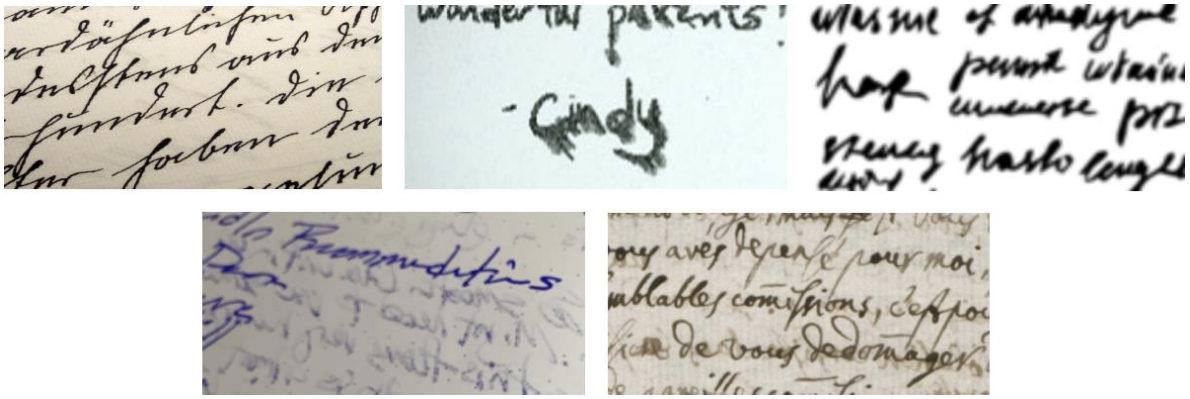

Рисунок 1.2 – Проблеми розпізнавання рукописного тексту

Аналіз цих викликів та проблем є важливим етапом дослідження предметної області, оскільки він допомагає зрозуміти складності, з якими стикаються вже існуючі методи та технології розпізнавання рукописного коду. Враховуючи ці проблеми, можна розробити більш ефективний та точний програмний додаток для розпізнавання рукописного програмного коду з використанням мови Python.

В області розпізнавання рукописного програмного коду проведено значну кількість досліджень та розробок,особливо протягом останніх кількох років, спрямованих на поліпшення точності, швидкості розпізнавання. Деякі з актуальних досліджень та робіт в цій області включають:

– Розробка алгоритмів розпізнавання: Вчені та дослідники працюють над розробкою нових алгоритмів та методів розпізнавання рукописного коду. Вони використовують різні підходи, такі як нейронні мережі, машинне навчання та глибинне навчання, для покращення якості розпізнавання та зниження помилок.

– Використання нейронних мереж: Нейронні мережі стали потужним інструментом у розпізнаванні рукописного коду. Дослідники використовують глибинні нейронні мережі, такі як згорткові нейронні мережі (CNN) та рекурентні нейронні мережі (RNN), для забезпечення високої точності розпізнавання та врахування контексту при аналізі рукописного коду.

– Використання методів обробки зображень: Дослідники використовують методи обробки зображень для покращення якості вхідних даних перед розпізнаванням. Вони застосовують фільтри для видалення шуму, методи бінаризації для перетворення зображень у двійковий формат, а також алгоритми сегментації для виділення окремих символів зі зображення.

Аналіз цих актуальних досліджень та робіт дозволяє отримати уявлення про сучасні методи та технології, які використовуються у розпізнаванні рукописного програмного коду. Це допомагає визначити прогалини та можливості для подальшого дослідження. Все що є у нашому інструментарії – це набори даних, які ми можемо показувати моделі, вказуючи їй як потрібно себе коригувати задля правильного результату. Завдяки цьому точність розпізнавання стане більшою.

1.2 Мови високого рівня, особливості та основні структурні рішення

В області розпізнавання рукописного програмного коду проведено значну кількість досліджень та розробок, спрямованих на поліпшення точності, швидкості розпізнавання. Деякі з актуальних досліджень та робіт в цій області включають:

1. Optical Character Recognition (OCR): OCR використовується для розпізнавання рукописного тексту шляхом аналізу форми та структури символів.

Вона має такі переваги:

– Висока швидкість розпізнавання, особливо для надрукованого тексту.

– Здатність розпізнавати різні стилі рукопису.

- Зручність використання та наявність готових рішень, таких як Tesseract OCR, що спрощують розробку та інтеграцію.

Проте OCR має деякі недоліки:

– Висока вразливість до неправильної чи незрозумілої рукописної форми.

– Погана продуктивність при розпізнаванні рукописного тексту з багатьма складними з'єднаннями та накладаннями символів.

– Обмежена точність в порівнянні з розпізнаванням надрукованого тексту.

2. Recurrent Neural Networks (RNN): RNN - це тип нейромережі, який добре справляється з аналізом послідовностей, таких як рукописний текст.

Вона має такі переваги:

– Здатність моделювати довгострокові залежності у послідовностях символів.

– Добре підходить для розпізнавання нерегулярного та складного рукописно-го тексту.

Проте RNN також має деякі недоліки:

– Вимагає значних обчислювальних ресурсів та тривалого навчання.

– Чутлива до шуму та сканованих артефактів на зображеннях.

– Вимагає великої кількості навчальних даних для досягнення хорошої точності.

Деякі з актуальних досліджень та робіт в цій області включають:

3. Long Short-Term Memory (LSTM): LSTM - це варіант RNN, що має поліпшену здатність запам'ятовування та управління довгостроковими залежностями.

Вона має такі переваги:

– Здатність ефективно моделювати залежності в рукописному тексті.

– Здатність запам'ятовувати контекст та інформацію з великого діапазону символів.

Недоліки LSTM схожі на недоліки RNN і включають вимогу до обчислювальних ресурсів, нестабільність у випадку шумних зображень та необхідність великої кількості навчальних даних. Це зовсім не годиться для наших поставлених задач

Рекурентні нейронні мережі (RNN) широко використовуються в розпізнаванні рукописного тексту і показують ефективні результати. Давайте розглянемо деталі цього методу.

RNN є спеціальним типом нейронних мереж, які здатні працювати з послідовними даними, такими як рукописний текст. Одна з ключових особливостей RNN полягає в тому, що вона зберігає внутрішній стан, який може передаватися з одного кроку до іншого. Це дозволяє RNN враховувати контекстуальну інформацію з попередніх символів під час розпізнавання поточного символу.

У контексті розпізнавання рукописного тексту, RNN може бути використана для послідовного розпізнавання окремих символів на основі попередніх символів. Мережа приймає послідовність вхідних зображень символів (наприклад, скановані зображення літер) і видає ймовірності для кожного можливого символу.

Одна з популярних архітектур RNN, що застосовується в розпізнаванні рукописного тексту, - Long Short-Term Memory (LSTM). LSTM має покращені механізми збереження та забування інформації, що робить її більш ефективною для моделювання довготривалих залежностей в послідовностях символів.

Ефективність RNN у розпізнаванні рукописного тексту залежить від кількох факторів, таких як якість вхідних зображень, розмір навчального набору даних, архітектура мережі та параметри навчання. Якщо забезпечені якісні та чіткі зображення символів, а також достатній обсяг навчальних даних, RNN може досягти високої точності в розпізнаванні рукописного тексту.

Загалом, RNN є потужним і ефективним інструментом для розпізнавання рукописного тексту, але його ефективність залежить від якості даних, обсягу навчального набору та правильного налаштування моделі.

Опис того, як працюють рекурентні нейронні мережі (RNN) в розпізнаванні рукописного в цій області тексту:

– Вхідні зображення символів конвертуються в підходящий формат (наприклад, вектори або матриці). Зображення можуть бути попередньо оброблені для поліпшення якості (налаштування контрастності, видалення

шуму тощо). Дані можуть бути розділені на тренувальний та тестовий набори.

– RNN складається з повторюваних блоків, які мають внутрішні стани. Кожен блок має вхідний шар, прихований шар та вихідний шар. Прихований стан блоку передається в наступний блок для врахування контексту.

– Ваги мережі ініціалізуються випадковими значеннями. Вхідні зображення подаються на вхід мережі, а вихідні символи порівнюються зі справжніми мітками. Використовуючи метод зворотного поширення помилки, здійснюється корекція ваг мережі. Процес навчання повторюється протягом кількох етапів, доки не досягнутий достатній рівень точності.

– Навчена RNN модель приймає на вхід нові зображення символів. Мережа видає ймовірності для кожного можливого символу. Використовуючи алгоритм декодування, визначається послідовність символів, що найбільш ймовірно відповідає рукописному тексту.

В цілому, RNN є ефективним методом для розпізнавання рукописного тексту, оскільки вони здатні враховувати контекст та залежності між символами. Однак, їх використання пов'язане з високими обчислювальними вимогами та потребою у великому обсязі навчальних даних для досягнення задовільної точності.

LSTM (Long Short-Term Memory) є одним з варіантів рекурентних нейронних мереж (RNN), який широко застосовується в розпізнаванні рукописного тексту. Він був розроблений для подолання проблеми зниклого градієнта, що виникає при тренуванні глибоких RNN.

LSTM має додаткові структурні одиниці, так звані "клітини пам'яті" (memory cells), які дозволяють зберігати та оновлювати інформацію на тривалий час. Вони складаються з трьох важливих компонентів: «входовий вентиль», «виходовий вентиль» та «забувальний вентиль». Кожен з цих компонентів регулює, яка інформація повинна бути збережена, оновлена або видалена.

У розпізнаванні рукописного тексту LSTM може використовуватись для послідовності обробки пікселів або векторів , що представляють рукописний текст. Вхідні дані подаються послідовно в LSTM, і він розуміє залежності між попередніми та поточними вхідними даними, використовуючи клітини пам'яті. Після обробки всієї послідовності вхідних даних LSTM може здати вихідні символи

символи або класи, які представляють розпізнаний рукописний текст.

Ефективність LSTM в розпізнаванні рукописного тексту може бути дуже високою. Вони добре справляються з обробкою послідовностей даних, таких як рукописний текст, завдяки здатності зберігати та використовувати контекстуальну інформацію. LSTM може розпізнавати складні залежності в рукописному тексті та враховувати довготривалу інформацію, що робить його ефективним методом для розпізнавання рукописного тексту.

Проте, ефективність LSTM залежить від різних факторів, таких як якість тренувального набору даних, розмір та складність мережі, використані гіперпараметри тощо. Існує також можливість перенавчання LSTM, коли він не досягає адекватної універсальності для розпізнавання інших видів рукописного тексту, які не входять у тренувальний набір даних. Тому, хоча LSTM є потужним і ефективним методом, його успішність в розпізнаванні рукописного тексту залежить від різних факторів і потребує належної настройки та оптимізації.

Також для розпізнавання рукописного тексту використовується OCR (Optical Character Recognition) - це технологія, яка використовується для автоматичного розпізнавання тексту, що міститься на фізичних або електронних документах, зображеннях або сканах. Основна мета OCR - перетворити надрукований або рукописний текст у машинно-читабельний формат, який може бути оброблений комп'ютером.

OCR використовується в різних сферах, включаючи автоматизацію офісних робіт, розпізнавання паспортів та документів, архівування інформації, дослідження тексту та багато іншого. Існує багато програмних рішень та сервісів, таких як Tesseract OCR, Microsoft Azure OCR та Google Cloud Vision OCR, які надають інструменти для реалізації OCR-функціональності у проектах.

Окремий варіант розпізнавання рукописного тексту, який є найкращим, може залежати від конкретного використання та потреб користувача.

Найкращий варіант розпізнавання рукописного тексту залежить від контексту застосування та особливостей тексту. OCR може бути практичним варіантом для друкованого тексту та структурованих документів, тоді як RNN та LSTM можуть бути більш ефективними для розпізнавання рукописного тексту з

урахуванням послідовності та контексту.

Порівняння LSTM, RNN та OCR в контексті розпізнавання рукописного тексту можна представити у вигляді таблиці, що містить відповідні характеристики кожного методу. Ось інформативна таблиця порівняння:

Таблиця 1.1 - Порівняння технологій розпізнавання тексту

Метод	Опис	Переваги	Недоліки
LSTM	Розширення RNN з довготривалою пам’яттю	Здатність моделювати довгі залежності	Складність навчання та налаштування моделі
RNN	Рекурентна нейронна мережа	Здатність моделювати послідо-вності	Обмежена здатність до запам’ято-вування довгих залежностей
OCR	Оптичне розпізнавання символів	Ефективне для друкованого та структуро-ваного тексту	Труднощі з розпізна-Ванням непростих шрифтів, рукопису та неструктуро-ваних документів

Ця таблиця надає загальний огляд характеристик та переваг кожного методу в контексті розпізнавання рукописного тексту. Важливо враховувати конкретні потреби та обмеження проекту при виборі відповідного методу.

Враховуючи проведене порівняння, можна зробити висновок, що для розпізнавання рукописного тексту найкращим варіантом є OCR (оптичне розпізнавання символів), а не рекурентні нейронні мережі і розширення RNN з довготривалою пам’яттю. За допомогою методів перед обробки ми зможемо виправити недоліки OCR. Основні аргументи, які підтверджують цей висновок, включають:

– Ефективність: OCR є добре вивіреною технологією, яка показує високу

точність та швидкість розпізнавання друкованого та структурованого тексту.

– Широкий спектр застосувань: OCR може застосовуватися для розпізнавання тексту в різних контекстах, включаючи документи, книги, статті, форми та інші типи друкованого тексту.

– Надійність: OCR відомий своєю стабільністю та надійністю, що робить його популярним варіантом для багатьох розпізнавальних завдань.

У той час як RNN та LSTM можуть бути корисними для моделювання послідовних залежностей, вони мають свої обмеження, такі як складність навчання та налаштування, а також обмежену здатність до запам'ятовування довгих залежностей. Тимчасово, OCR має більшу перевагу у розпізнаванні рукописного тексту, оскільки він спеціально розроблений для роботи з символами та шрифтами.

Основна мета OCR полягає в тому, щоб комп'ютер може розпізнати та інтерпретувати текст, аналогічно до того, як це робить людина. Це важливо для автоматизації обробки великих обсягів текстових даних, таких як документи, книги, газети, форми та інші джерела, що містять друкований текст.

Процес OCR може бути розділений на кілька етапів. Починаючи з попередньої обробки зображення, яка включає виправлення спотворень, фільтрацію шуму та нормалізацію зображення. Це важливо для покращення якості зображення та поліпшення подальших етапів OCR.

Після попередньої обробки зображення відбувається виявлення текстових областей. Це включає алгоритми сегментації зображення, які допомагають визначити місцезнаходження та розміри текстових блоків на зображенні.

Далі розпочинається фаза розпізнавання символів. На цьому етапі текстові блоки розбиваються на окремі символи або слова, і проводиться класифікація цих символів за допомогою моделей машинного навчання, таких як нейронні мережі, приховані моделі Маркова (HMM) або інші алгоритми розпізнавання. Ключовим аспектом цього етапу є створення моделей, які можуть відповідати різним видам шрифтів, розмірів та стилів написання.

Один з ключових аспектів OCR - це здатність моделей використовувати

контекстуальну інформацію для поліпшення розпізнавання. Наприклад, модель може використовувати контекст навколо символу або слова для кращого розуміння його смислу та поліпшення точності розпізнавання. Одним з підходів, що використовуються для цього, є використання рекурентних нейронних мереж (RNN) і довготривалих залежних пам'ятей (LSTM), які можуть зберігати і використовувати контекстуальну інформацію при розпізнаванні символів.

Іншим важливим аспектом OCR є підтримка розпізнавання різних мов та алфавітів. Моделі OCR повинні бути навчені розпізнавати символи різних алфавітів, що включають латиницю, кирилицю, арабські літери, китайські ієрогліфи та багато інших. Це представляє технічні виклики в області OCR, оскільки кожен алфавіт має свої унікальні особливості та вимагає окремого навчання та адаптації моделей.

У підсумку, OCR є важливою технологією для автоматичного розпізнавання тексту з зображень. Вона використовує різні етапи, включаючи попередню обробку, виявлення тексту та розпізнавання символів з використанням моделей машинного навчання. Однак, точність OCR може залежати від якості зображення, типу шрифту та інших факторів. Технології, такі як RNN та LSTM, можуть покращити точність розпізнавання, використовуючи контекстуальну інформацію та залежності між символами. Розвиток OCR продовжується, і в майбутньому можна очікувати подальшого поліпшення точності та можливостей розпізнавання рукописного тексту.

Враховуючи ці фактори, OCR вважається найкращим варіантом для розпізнавання рукописного тексту, забезпечуючи надійність та ефективність у багатьох сценаріях застосування.

Все ж, варто також відзначити глибокі згорткові нейронні мережі (Convolutional Neural Networks, CNN) в розпізнаванні рукописного тексту.Ці нейронні мережі є потужними інструментами в області розпізнавання тексту. Вони використовуються для автоматичного виявлення та класифікації текстової інформації на зображеннях. CNN широко застосовуються в задачах розпізнавання рукописного тексту, OCR та інших схожих завданнях.

Основними складовими CNN є згорткові шари (Convolutional layers) та

пулінгові шари (Pooling layers). Згорткові шари використовуються для виявлення локальних особливостей та характеристик у вхідних даних. Кожен шар складається з набору фільтрів (ядер), які здійснюють згортку з вхідними даними для виділення особливостей. Результатом згортки є карти ознак (feature maps), які містять інформацію про виявлені особливості на різних рівнях деталізації.

Пулінгові шари використовуються для зменшення розмірності карти ознак та витягування найважливіших характеристик. Зазвичай застосовуються максимальне пулінгування (Max pooling), при якому вибирається найбільше значення з кожного регіону карти ознак. Це дозволяє зберегти важливу інформацію та зменшити кількість параметрів моделі.

Окрім згорткових і пулінгових шарів, CNN може включати повністю зв'язані шари (Fully Connected layers) для класифікації результатів. Повністю зв'язані шари приймають векторизовані ознаки з попередніх шарів та здійснюють класифікацію на основі навчених ваг.

Одним із викликів у використанні CNN є потреба у великій кількості анотованих даних для навчання. Для досягнення високої точності моделі необхідно мати значну кількість зображень тексту для тренування. Крім того, обробка зображень вимагає значних обчислювальних ресурсів, особливо при використанні глибоких та складних мереж.

У певних сценаріях, де розпізнавання тексту здійснюється на структурованому текстовому матеріалі, традиційні методи OCR можуть бути ефективними. Вони можуть забезпечити високу точність та швидкість розпізнавання. Однак, у випадках, коли рукописний текст неструктурований та має складну синтаксичну структуру, CNN разом зі своєю здатністю до виявлення шаблонів та характеристик зображення може бути більш ефективним підходом. Все ж, за допомогою методів перед обробки це можна виправити.

Отже, CNN є потужним інструментом у розпізнаванні тексту, зокрема рукописного. Вони дозволяють автоматично виявляти та класифікувати текстову інформацію на зображеннях. Використання CNN може забезпечити високу точність розпізнавання, особливо у складних сценаріях з неструктурованим рукописним текстом.

1.3 Програмні засоби розпізнавання рукописного тексту

На сьогоднішній день існує безліч програмних засобів OCR, які пропонуються різними компаніями та дослідницькими групами. У цьому пункті ми порівняємо декілька з них: Tesseract OCR, ABBYY FineReader, Microsoft Azure OCR, Google Cloud Vision OCR та Adobe Acrobat OCR. Надаємо детальний огляд кожного засобу та порівнюємо їх за різними аспектами.

В порівнянні з Tesseract OCR, ABBYY FineReader є комерційним програмним засобом, який також відомий своїми високими можливостями розпізнавання тексту. ABBYY FineReader пропонує розширений набір інструментів для оптимізації результатів розпізнавання, таких як виправлення помилок, визначення структури документа та імпорт/експорт у різних форматах. Він також підтримує багато мов та має можливості розпізнавання рукописного тексту.

Microsoft Azure OCR - це хмарний сервіс, розроблений Microsoft, який надає широкий спектр можливостей розпізнавання тексту. Він підтримує розпізнавання різних мов, включаючи рідкісні алфавіти та символи. Azure OCR також має вбудовану підтримку розпізнавання рукописного тексту та дозволяє використовувати інформацію для покращення результатів розпізнавання. Microsoft Azure відомий своєю високою точністю й швидкістю.

Google Cloud Vision OCR - це інший хмарний сервіс, розроблений Google, який пропонує потужні можливості розпізнавання тексту. Він має високу точність розпізнавання та здатність працювати з різноманітними типами зображень. Cloud Vision OCR підтримує багатомовне розпізнавання та може розпізнавати рукописний текст.

У порівнянні з цими програмними засобами, Tesseract OCR відрізняється своєю відкритістю та активною спільнотою розробників. Він може бути більш гнучким і налаштовуваним для конкретних потреб. Тим не менше, комерційні засоби, такі як ABBYY FineReader, Microsoft Azure OCR, Google Cloud Vision OCR та Adobe Acrobat OCR, зазвичай мають розширений функціонал.

Таблиця 1.2 – Порівняння програмних засобів OCR

Система OCR	Швидкість	Точність	Доступність	Інтеграція
Tesseract OCR	середня	висока	Відкрите Джерело	API,SDK
Google Cloud Vision OCR	висока	висока	Хмарна Платформа	API
Microsoft Azure OCR	висока	висока	Хмарна Платформа	API
ABBYY FineReader	висока	висока	Комерційний	SDK
Adobe Acrobat OCR	висока	висока	Комерційний	ПЗ

Ця таблиця надає загальний огляд популярних OCR систем та їхніх характеристик. Важливо враховувати, що точність та швидкість розпізнавання можуть варіюватися залежно від конкретних сценаріїв застосування, які включають різні типи документів та рівень складності тексту. Крім того, доступність та рівень інтеграції також можуть бути важливими факторами при виборі OCR системи для конкретного проекту.

Огляд існуючих програмних засобів OCR свідчить про те, що ця технологія широко застосовується у багатьох галузях, включаючи документообіг, архівування, автоматизацію бізнес-процесів, медичний сектор, фінансову сферу та інші. Вона дозволяє підвищити ефективність та точність обробки текстової інформації, зменшити зусилля, необхідні для ручного введення даних та зберегти час.

Зважаючи на постійний розвиток методів машинного навчання, нейромереж та комп'ютерного зору, можна очікувати подальше вдосконалення програмних засобів OCR, зокрема покращення точності розпізнавання, розширення функціональності та підтримка нових типів документів.

1.4 Постановка задач кваліфікаційної роботи та шляхи її вирішення

Постановка задач кваліфікаційної роботи та шляхи її вирішення у темі "Розпізнавання рукописного програмного коду мовою Python" є ключовим для визначення мети та завдань роботи, а також стратегії для їх досягнення. Детально розкриємо цей пункт.

Для досягнення цієї мети необхідно вирішити наступні завдання:

1) Проаналізувати основні характеристики цифрових зображень та сфери їх використання

2) Провести аналіз особливостей мов високого рівня на прикладі мови програмування Python.

3) Проаналізувати існуючі програмні рішення по розпізнаванню цифрових зображень, виділити їх основні структурні елементи;

4) Дослідити алгоритми розпізнавання рукописної інформації на цифрових зображеннях.

5) Розробити розпізнавання рукописного тексту та трансляції його в програмний код.

6) Провести тестування та порівняльний аналіз реалізованого програмного додатку розпізнавання рукописного програмного коду мовою Python.

Шляхи вирішення задач:

– Використання передових методів обробки зображень для покращення якості вхідних даних, зокрема застосування фільтрації шуму, нормалізації контрасту та розмиття.

– Вибір підходящої OCR-системи, яка підтримує розпізнавання рукописного тексту та має високу точність.

– Валідація та оцінка продуктивності розробленої системи на тестовому наборі даних, порівняння з результатами інших OCR-систем.

2. МЕТОДИ, АЛГОРИТМИ ОБРОБКИ ТА РОЗПІЗНАВАННЯ РУКОПИСНИХ ТЕКСТІВ

2.1 Алгоритми попередньої обробки цифрових зображень рукописних текстів

Алгоритми передобробки для розпізнавання рукописного тексту є важливою складовою в системі розпізнавання рукописного програмного коду мовою Python та Tesseract OCR. Їх метою є покращення якості зображення та виділення символів для поліпшення точності розпізнавання.

Адаптивне порогове бінаризування і звичайне порогове бінаризування є двома підходами до перетворення зображення у бінарне, де пікселі поділяються на дві категорії - фон та об'єкт. Порівняємо їх технічні аспекти:

Звичайне порогове бінаризування:

– Вимагає задання глобального порогу для всього зображення.

– Кожен піксель зображення порівнюється з цим глобальним порогом.

– Якщо значення пікселя більше порогового значення, воно встановлюється на максимальне значення (наприклад, 255) - об'єкт.

– Якщо значення пікселя менше порогового значення, воно встановлюється на мінімальне значення (наприклад, 0) - фон.

Адаптивне порогове бінаризування:

– Використовується локальний поріг, що змінюється в залежності від контексту пікселя.

– Розділяє зображення на малих локальних блоках або шаблонах.

– Для кожного блоку обчислюється локальний поріг, що базується на статистичних характеристиках блоку, наприклад, середньому значенні або середньому значенні плюс деяка константа.

– Кожен піксель блоку порівнюється з локальним порогом блоку.

– Застосовується бінаризація, аналогічна звичайному пороговому бінаризуванню, на основі локального порогу.

Звичайне порогове бінаризування використовує глобальний поріг для всього зображення, що може бути ефективним, якщо освітлення однорідне. Проте, воно може бути недостатньо ефективним у випадках, коли освітлення

Адаптивне порогове бінаризування дозволяє враховувати локальні властивості зображення та адаптувати поріг для кожного блоку окремо. Це дозволяє краще розділяти об'єкти в умовах нерівномірного освітлення або змінюючогося контрасту. Якщо не застосовувати адаптивне порогове бінаризування (рисунок 2.1,с) і використовувати лише звичайне порогове бінаризування (рисунок 2.1, б), то можуть виникнути проблеми. Велика частина зображення може бути неправильно класифікована через зміну освітлення або контрасту. Малі об'єкти або деталі можуть бути втрачені, оскільки глобальний поріг не враховує їх специфіку. Є висока ймовірність отримання забруднених або неправильно збінаризованих контурів символів, що може погіршити якість розпізнавання рукописного коду мовою Python та Tesseract OCR. Тому, для теми "Розпізнавання рукописного програмного коду мовою Python", застосування адаптивного порогового бінаризування буде корисним, оскільки воно дозволяє краще адаптуватися до особливостей зображень з рукописним кодом, покращує роздільну здатність символів та зменшує кількість помилок під час розпізнавання.

а) б) с)

Рисунок 2.1 – Порівняння застосування звичайного порогового бінаризування й адаптивного

Адаптивне порогове бінаризування є методом обробки зображень, який дозволяє розділити зображення на дві категорії - чорний і білий - залежно від яскравості пікселів. Основна мета цього алгоритму полягає у встановленні

оптимальних порогових значень для кожного пікселя або його околиці залежно від локальних характеристик зображення. Такий підхід дозволяє покращити якість бінаризації і зберегти деталі об'єктів на зображенні.

Починаючи з початкового зображення, розміром M на N пікселів, область розгляду пересувається по всьому зображенню з кроком, який визначається заздалегідь. Для кожної окремої області розгляду визначається локальне порогове значення. Це порогове значення може бути обчислене за допомогою різних методів, наприклад, на основі середнього арифметичного або методу Оцу. Основна ідея полягає в тому, щоб встановити поріг таким чином, щоб об'єкти, які мають яскравість нижчу за поріг, вважалися чорними, а об'єкти з яскравістю вище порогу - білими. Після визначення порогового значення для кожної області розгляду, кожен піксель в цій області порівнюється з відповідним порогом. Якщо яскравість пікселя менше порогового значення, піксель вважається чорним, в іншому випадку - білим. Цей процес виконується для кожної області розгляду на всьому зображенні, утворюючи нове бінаризоване зображення. Результатом адаптивного порогового бінаризування є зображення, в якому об'єкти відокремлені від фону шляхом чіткого контрасту між чорним і білим. До теми "Розпізнавання рукописного програмного коду мовою Python та Tesseract OCR" алгоритм адаптивного порогового бінаризування може бути корисним на етапі підготовки зображень перед подальшим розпізнаванням. Його застосування дозволить відділити символи від фону, створити чіткі контури та поліпшити якість розпізнавання. Завдяки своїй здатності адаптуватись до локальних особливостей зображення, алгоритм адаптивного порогового бінаризування може забезпечити кращі результати для рукописного тексту.

Інший варіант алгоритму препроцесингу - це фільтрація за допомогою медіанного фільтра. Цей алгоритм застосовується для зменшення шуму на зображенні. Він обробляє кожен піксель шляхом заміни його значення медіаною значень пікселів у визначеній області навколо нього. Медіанний фільтр ефективно видаляє шум, зберігаючи при цьому краєві та інші важливі деталі символів.

Медіанний фільтр є одним з методів фільтрації зображень, який дозволяє

зменшити шум і поліпшити якість зображення. Його особливість полягає в тому, що він використовує медіанне значення пікселів у визначеній околиці для заміни значення пікселя, що фільтрується.

Опис роботи медіанного фільтра:

– Починаємо з початкового зображення, розміром M на N пікселів.

– Обираємо розмір маски фільтра, який визначає кількість пікселів, що використовуються для обчислення медіанного значення. Зазвичай маска має квадратну форму і зазвичай використовується непарне число пікселів для забезпечення наявності центрального пікселя.

– Проходимо кожен піксель зображення і для кожного пікселя застосовуємо медіанний фільтр.

– Для кожного пікселя встановлюється значення, що відповідає вибраному розміру маски фільтра. Значення пікселів в цій околиці сортуються за зростанням.

Медіанне значення обчислюється шляхом вибору середнього значення відсортованих пікселів. Якщо кількість пікселів в масці непарна, то медіана буде значенням, яке розташоване посередині. Якщо кількість пікселів парна, то медіана обчислюється як середнє значення двох пікселів, розташованих в центрі. Далі заміняємо значення фільтрованого пікселя на обчислене медіанне значення. Продовжуємо цей процес для кожного пікселя на всьому зображенні, утворюючи нове відфільтроване зображення. Медіанний фільтр є ефективним у видаленні шуму, особливо коли шум має артефакти. Він добре зберігає краї та деталі об'єктів на зображенні, оскільки не враховує значення викинутих або вкраплених пікселів, а використовує лише медіанне значення. Щодо його застосування до теми "Розпізнавання рукописного програмного коду мовою Python та Tesseract OCR", медіанний фільтр може бути корисним на етапі попередньої обробки зображень перед подальшим розпізнаванням. Він допоможе зменшити шум та поліпшити контури символів, що забезпечить кращу якість розпізнавання коду.

На рисунку 2.2 наведено блок-схему алгоритму фільтрації за допомогою медіанного фільтра. Цей алгоритм використовується для зменшення шуму та

видалення артефактів на зображеннях. Він ефективно розмиває точки, які відрізняються від більшості пікселів у своєму навколишньому середовищі, допомагаючи згладити деталі та покращити якість зображення

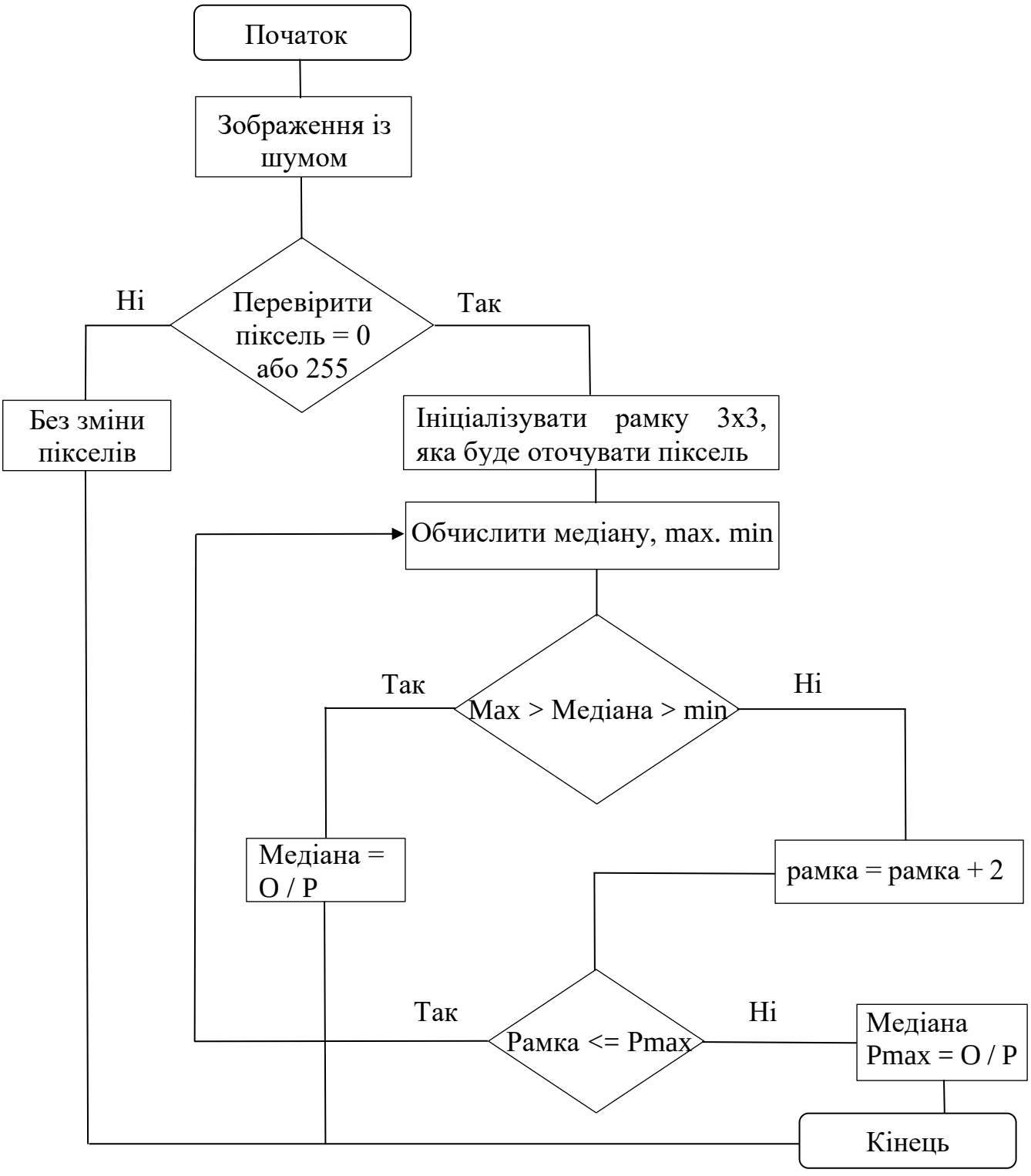

Рисунок 2.2 - Блок-схема алгоритму фільтрації за допомогою медіанного фільтра

Крім того, важливо застосовувати алгоритми розширення та скорочення

контуру. Алгоритм розширює контур символу шляхом використання морфологічних операцій, таких як розширення та ерозія. Основна ідея полягає у застосуванні математичних операцій на пікселях зображення для збільшення або зменшення їх відмітності, в залежності від їхнього оточення. Це допомагає заповнити пропущені області в символах та зробити їх більш компактними. З іншого боку, алгоритм скорочення контуру видаляє непотрібні шуми та зайві деталі, зберігаючи при цьому основну структуру символу.Застосування алгоритму розширення та скорочення контуру до зображень з текстом допомагає покращити чіткість та контрастність контурів символів. Це може полегшити процес розпізнавання рукописного тексту, дозволяючи алгоритмам OCR (наприклад, Tesseract OCR) краще впізнавати та інтерпретувати символи зображеного тексту.

З урахуванням особливостей рукописного програмного коду мовою Python та Tesseract OCR, найкращим алгоритмом препроцесингу буде комбінація адаптивного порогового бінаризування та фільтрації медіанним фільтром. Це дозволить ефективно виділяти символи, зменшуючи вплив шуму та зберігаючи важливі деталі. Додатково, застосування алгоритмів розширення та скорочення контуру допоможе вирізняти символи на зображенні та покращити якість розпізнавання рукописного коду.

2.2 Алгоритми розпізнавання об’єктів на цифрових зображеннях

Сегментація та виділення ознак рукописного тексту - це процеси, які дозволяють визначити окремі символи або слова у зображенні рукописного тексту для подальшого аналізу та розпізнавання. Розглянемо ці процеси більш детально:

Сегментація полягає у визначенні границь окремих символів або слів на зображенні рукописного тексту. Цей процес може бути виконаний за допомогою різних методів, таких як:

a) Порогова сегментація: Зображення перетворюється на дві категорії - фон та символи, шляхом використання певного порогового значення. Пікселі з інтенсивністю, меншою за порогове значення, вважаються фоном, а ті, що більше або рівні пороговому значенню, вважаються символами.

b) Методи на основі графів: Зображення може бути перетворене на граф, де пікселі представлені як вершини, а зв'язки між пікселями - як ребра. Застосовуючи алгоритми обходу графа, можна визначити окремі компоненти, що відповідають символам.

c) Методи на основі контурів: Застосування алгоритмів пошуку контурів дозволяє виділити границі символів. Після цього можна провести операції сегментації, використовуючи ці контури.

Порогова сегментація - це метод сегментації зображення, при якому пікселі розділяються на дві категорії: фон та об'єкти (у нашому випадку символи рукописного тексту). Основна ідея полягає у використанні певного порогового значення, яке розділяє пікселі на дві групи в залежності від їхньої інтенсивності. Пікселі з інтенсивністю, меншою за порогове значення, вважаються фоном, а пікселі з інтенсивністю, більшою або рівною пороговому значенню, вважаються символами тексту. Порогова сегментація є простим і популярним методом для виділення об'єктів на зображенні шляхом встановлення порогового значення для пікселів. Пікселі зі значеннями, які перевищують поріг, вважаються об'єктами, тоді як пікселі зі значеннями, які менші або рівні порогу, вважаються фоном. Порогова сегментація – мабуть найефективніший метод передобробки.

Математично, порогова сегментація може бути виражена наступним чином:

Для кольорових зображень:

1. Нехай I(x, y) буде кольоровим зображенням з координатами пікселів (x, y).
2. Нехай T буде пороговим значенням.

Тоді сегментація може бути виконана наступним чином:

– Якщо $I(x, y) > T$, тоді встановити піксель (x, y) як об'єкт.
– Якщо $I(x, y) \leq T$, тоді встановити піксель (x, y) як фон.

Для зображень в градаціях сірого:

1. Нехай *I(x, y)* буде зображенням в градаціях сірого з координатами пікселів (x, y).
2. Нехай T буде пороговим значенням.

Тоді сегментація може бути виконана наступним чином:

– Якщо $I(x, y) > T$, тоді встановити піксель (x, y) як об'єкт.

– Якщо $I(x, y) \leq T$, тоді встановити піксель (x, y) як фон.

Приклад, який використовує надані математичні формули для порогової сегментації та ілюструє її результат на графіку зображений на рисунку 2.3.

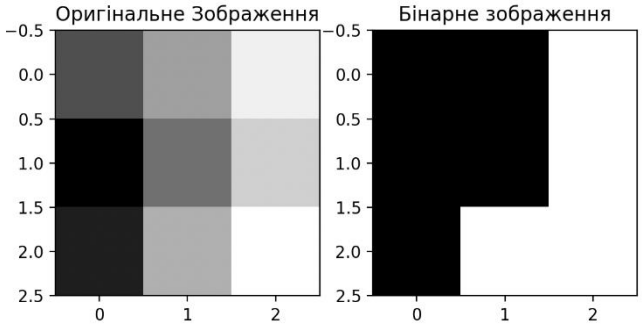

Рисунок 2.3 – Приклад виконання порогової сегментації

У цьому прикладі ми створюємо масив I, який представляє зображення в градаціях сірого. Встановлюємо порогове значення T і виконуємо порогову сегментацію шляхом порівняння кожного пікселя з пороговим значенням. Якщо піксель перевищує поріг, встановлюємо його значення як 1 (об'єкт), в іншому випадку встановлюємо 0 (фон). На графіку відображаються оригінальне та сегментоване зображення.

Це математичне вираження показує, що порогова сегментація залежить від порогового значення, яке вибирається для відокремлення об'єктів від фону на зображенні. Оптимальне значення порогу може варіюватися в залежності від особливостей конкретного зображення та задачі сегментації.

Іншим методом сегментації - метод на основі графів, який є одним із підходів до розподілу об'єктів на зображенні на основі аналізу графової структури. Цей метод може бути застосований до розпізнавання рукописного коду мовою Python.

Одним з найпоширеніших методів на основі графів є алгоритм знайомий як "Graph Cut" або "Графічний розріз". Ідея цього методу полягає в тому, що зображення розглядається як граф, де пікселі представлені вершинами, а зв'язки між ними відображають схожість пікселів. Завдання полягає в розбитті графу на дві частини - об'єкт та фон, шляхом зрізання деяких зв'язків.

Математично, порогова сегментація може бути виражена наступним чином:

– Нехай G(V, E) буде графом, де V - множина вершин (пікселів), E - множина зв'язків між вершинами.

– Нехай W(i, j) буде вагою зв'язку між вершинами i та j, що відображає схожість між пікселями.

– Нехай S і T будуть підмножинами вершин G, що відповідають об'єкту та фону відповідно.

Математично, порогова сегментація може бути виражена наступним чином: minimize $\sum_{[i \in S,\, j \in T]}$ W(i, j) де $[i \in S,\, j \in T]$ - сума по всіх зв'язках, які перетинають між множинами S та T.

Алгоритм графічного розрізу використовує різні підходи для знаходження мінімального розрізу, такі як алгоритм Бойковського-Колмогорова або алгоритм Ford-Fulkerson. Ці алгоритми базуються на теорії максимального потоку та мінімального розрізу в графах.

Тепер щодо питання, чи підійде цей метод для розпізнавання рукописного коду мовою Python. Графічний розріз може бути використаний для сегментації об'єктів на зображенні, включаючи рукописний код. Проте, варто враховувати, що ефективність методу може залежати від різних факторів, таких як складність зображення, розмір та складність графу. Рукописний код може мати велику варіативність форм та стилів написання, що може ускладнити сегментацію. Потрібно провести відповідні експерименти та оптимізації для досягнення задовільних результатів при використанні цього методу для розпізнавання рукописного коду мовою Python.

На жаль, контурна сегментація може бути менш ефективною для розпізнавання тексту, особливо якщо текст має низьку контрастність або існують

шуми або інші перешкоди на зображенні. Метод на основі контурів зазвичай використовується для виявлення форм та об'єктів загального характеру, а не для точного розпізнавання тексту. На рисунку 2.4 наведений приклад, який порівнює метод на основі графів (за допомогою алгоритму Graph Cut), метод порогової сегментації і метод на основі контурів.

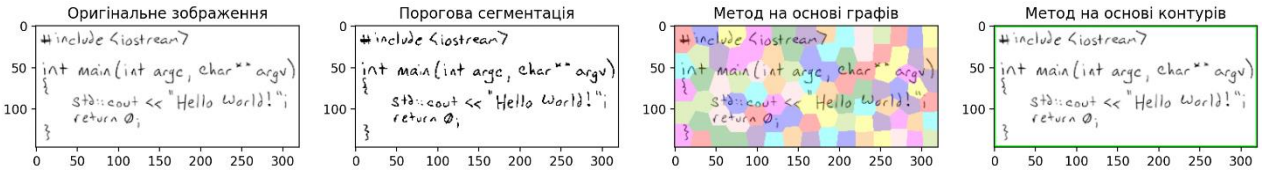

Рисунок 2.4 – Приклад порівняння різних алгоритмів сегментації тексту

Ось загальний алгоритм сегментації для розпізнавання рукописного коду за допомогою мови програмування Python та бібліотеки Tesseract OCR:

– Завантаження рукописного зображення з кодом.

– Виконання попередньої обробки зображення (якщо необхідно), таку як зменшення шуму, розмиття або підвищення контрастності.

– Використання методу порогової сегментації для розділення зображення на окремі символи.

– Виконання попередньої обробки до кожного символу.

– Використання Tesseract OCR для розпізнавання символу та отримання текстового відповідника.

Цей алгоритм може бути деталізованим та настроєним залежно від конкретних вимог та властивостей рукописного коду.

2.3 Алгоритм розпізнавання та трансляції рукописного тексту в код

Алгоритм розпізнавання рукописного програмного коду може бути корисним і потужним інструментом, він також має свої обмеження та виклики, пов'язані з точністю розпізнавання, обробкою складних структур, мовною

підтримкою та необхідністю настройки параметрів. Алгоритм може бути адаптований та розширений залежно від конкретних потреб та вимог.

Запропонований алгоритм містить наступні етапи роботи:

Етап 1. Завантаження зображення: Зчитування вхідного зображення, на якому міститься рукописний код, у форматі, який підтримується бібліотекою Python (зображення у форматі JPEG або PNG).

Етап 2. Застосування фільтрів для зменшення шуму на зображенні, таких як фільтр Гауса або медіанний фільтр. Це допоможе згладити зображення та видалити непотрібні дрібні деталі або шуми.

Етап 3. Якщо на зображенні присутній фон або непотрібні об'єкти, можна застосувати методи видалення фону, такі як порогова сегментація або методи видалення фону на основі текстури.

Етап 5. Покращення контрастності зображення для кращої видимості тексту. Це може бути досягнуто за допомогою методів, таких як гістограма розтягнення, адаптивне підсилення контрасту або еквалізація гістограми.

Етап 6. Перетворення зображення в двійкове зображення, де текст відокремлюється від фону. Це можна зробити шляхом встановлення певного порогового значення, яке відокремлює пікселі тексту від пікселів фону

Етап 7. Виявлення нахилу тексту та його виправлення для забезпечення горизонтального розташування тексту. Це можна здійснити за допомогою методів, таких як виявлення кута нахилу за допомогою лінійного перетворення Хафа та афінного перетворення.

Етап 8. Використання алгоритмів комп'ютерного зору та обробки зображень для виявлення окремих блоків коду на зображенні. Це може включати виявлення контурів, сегментацію за кольором або інші методи визначення регіонів інтересу.

Етап 9. Вирізання окремих блоків коду з вихідного зображення та збереження їх у окремих зображеннях для подальшого розпізнавання.

Етап 10. Застосування Tesseract OCR або іншої відповідної OCR бібліотеки для розпізнавання символів у кожному блоку коду. OCR аналізує зображення символу та спробує визначити його відповідний текстовий еквівалент.

Етап 11. Збір розпізнаних символів та їх позицій в кожному блоку коду для подальшого використання або аналізу.

Етап 12. Застосування додаткових алгоритмів післяобробки для поліпшення точності та якості розпізнавання. Це може включати корекцію помилок OCR, фільтрацію результатів, перевірку на правопис тощо.

Цей алгоритм поетапно обробляє рукописний код, використовуючи методи обробки зображень та OCR для отримання текстового відображення коду. Він дозволяє автоматизувати процес розпізнавання та аналізу рукописного програмного коду, що спрощує подальшу обробку, аналіз та інтерпретацію коду. Графічно запропонований алгоритм можна представити у вигляді наступної блок-схеми (рисунок 2.5):

Рисунок 2.5 – Блок-схема процесу розпізнавання та аналізу рукописного програмного коду

До переваг розробленого підходу такі:

– Алгоритм може працювати з різними типами рукописного коду та різними мовами програмування. Це дозволяє застосовувати його в різних сценаріях та проектах.

– Алгоритм може бути автоматизованим, що дозволяє швидко та ефективно обробляти великий обсяг рукописного коду без необхідності ручної роботи.

– Застосування Tesseract OCR та інших методів розпізнавання тексту дозволяє досягти високої точності розпізнавання, зменшуючи кількість помилок.

– Розпізнаний текст може бути використаний для подальшого аналізу, автоматичної перевірки синтаксису, генерації документації, аналізу залежностей та інших задач пов'язаних з програмним кодом.

– Використання автоматичного розпізнавання рукописного коду може значно скоротити час, необхідний для аналізу та обробки програмного коду, що забезпечує більш ефективну роботу розробників.

– Алгоритм може бути застосований до великих обсягів рукописного коду без втрати точності та продуктивності.

Серед недоліків є:

– Якщо рукописний код містить складні структури, такі як вкладені цикли, умовні оператори або функції, можуть виникати проблеми з правильним розпізнаванням цих структур. Це може призвести до неправильного визначення меж блоків коду або їх неправильного інтерпретації.

– Деякі OCR системи, зокрема Tesseract OCR, можуть мати обмежену підтримку деяких мов програмування або спеціальних символів, що використовуються у коді. Це може обмежити застосування алгоритму для розпізнавання певних типів рукописного коду.

Загалом, такий алгоритм розпізнавання рукописного програмного коду відкриває широкі можливості для автоматизації та поліпшення роботи з рукописним кодом, що дозволяє збільшити продуктивність.

3. ПРОГРАМНИЙ ДОДАТОК РОЗПІЗНАВАННЯ РУКОПИСНОГО ПРОГРАМНОГО КОДУ МОВОЮ PYTHON

3.1 Структура програмного додатку обробки цифрових зображень

У даному розділі наводяться основні відомості про технічне, інформаційне та інші види забезпечення, необхідні для розробки програмного забезпечення. Також, наводиться перелік модулів програмного забезпечення з зазначенням їхніх взаємозв'язків і обґрунтуванням виділення кожного з них. Обгрунтовується призначення й опис основних функцій для кожної частини програмного забезпечення.

У розробці системи для розпізнавання рукописного програмного коду з використанням мови Python та Tesseract OCR було вирішено відмовитися від мови програмування C++ та середовища розробки Microsoft Visual Studio 2019. Замість цього, ми використовували Python разом із бібліотекою OpenCV, яка надає можливості комп'ютерного зору та обробки зображень.

Мова програмування Python є високорівневою мовою, яка дозволяє швидко розробляти програмне забезпечення. Вона є популярним вибором для розробки різних застосунків, включаючи обробку зображень. Tesseract OCR є потужним інструментом для розпізнавання тексту, здатним працювати з рукописними символами.

OpenCV, або Open Source Computer Vision Library, є бібліотекою комп'ютерного зору з відкритим вихідним кодом. Вона надає набір функцій для обробки зображень, включаючи розпізнавання образів, роботу з відео та виявлення об'єктів. Завдяки своїм можливостям, OpenCV дозволяє нам реалізувати алгоритми розпізнавання рукописного програмного коду. OpenCV має відкриту ліцензію, яка дозволяє використовувати її у комерційних додатках. Це привертає велику спільноту користувачів, включаючи великі компанії, такі як IBM, Microsoft, Intel, Sony, Google, а також науково-дослідні центри, наприклад, Стенфорд, Массачусетський технологічний інститут, CMU та Кембридж.

У структурі нашого програмного додатку ми використали модульний

підхід, що дозволяє розбити програму на окремі модулі, які реалізують конкретні функції. Кожен модуль виконує свою задачу, працюючи з вхідними даними та власними функціями. Це спрощує розробку та підтримку коду. Таким чином, ми замінили мову програмування C++ на Python та Microsoft Visual Studio 2019 на OpenCV, аби розробити систему для розпізнавання рукописного програмного коду. Завдяки цим змінам, ми отримали зручне та потужне середовище для обробки зображень та виявлення тексту.

Для ефективної розробки програмного забезпечення для розпізнавання рукописного програмного коду мовою Python та використання Tesseract OCR рекомендується наступне технічне забезпечення:

– Операційна система: Windows 10 Professional (версія 10.0.19042). Вибір даної операційної системи обґрунтовується її поширеністю та сумісністю з необхідними інструментами та бібліотеками, що використовуються у проекті.

– Процесор: Intel Core i7-9700K, 3.60 ГГц, 8 ядер. Обрано потужний процесор з достатньою кількістю ядер для швидкої обробки зображень та виконання алгоритмів розпізнавання.

– Оперативна пам'ять (RAM): 16 ГБ DDR4. Достатній обсяг оперативної пам'яті для ефективної роботи з багатозадачними процесами та обробки великих обсягів даних.

– Простір на жорсткому диску: SSD, ємність 512 ГБ. Швидкий SSD-диск забезпечує швидкий доступ до даних та прискорює завантаження та збереження зображень та програмного коду.

Для ефективної розробки рекомендуються наступні інструменти розробки:

– Python: використовується Python 3.9.6. Python є основною мовою програмування для реалізації модулів та алгоритмів, пов'язаних з розпізнаванням рукописного коду та обробкою зображень.

– Tesseract OCR: використовується бібліотека Tesseract OCR 4.1.1. Tesseract OCR забезпечує можливість розпізнавання тексту з оброблених зображень та є ключовим інструментом для розпізнавання рукописного коду.

– OpenCV: використовується бібліотека OpenCV 4.5.3. OpenCV надає

потужні фвіфункції для обробки зображень, такі як порогове перетворення, корекція нахилу, адаптивне порогове перетворення та морфологічні операції.

Також варто перерахувати та описати необхідне інформаційне забезпечення, яке використовується для розробки програмного забезпечення з розпізнавання рукописного програмного коду Python за допомогою бібліотеки Tesseract OCR.

– Офіційна документація Tesseract OCR: Документація забезпечує детальний опис функцій, методів та можливостей Tesseract OCR, необхідних для розпізнавання тексту з оброблених зображень.

– Документація бібліотеки OpenCV: В офіційній документації OpenCV міститься інформація про функції та методи для обробки зображень, що використовуються у модулі обробки зображень.

– Документація Python: Офіційна документація Python містить вичерпну інформацію про мову програмування Python, стандартні бібліотеки та функціональні можливості, що використовуються у розробці.

Для розробки програмного забезпечення "розпізнавання рукописного програмного коду мовою Python та Tesseract OCR" виділено наступні модулі:

– Модуль передобробки зображення (Image Preprocessing Module) Призначення: Виконує передобробку зображень для підготовки до подальшого розпізнавання тексту. Функції: Вирівнювання зображень, порогове перетворення, корекція нахилу, видалення шуму.

– Модуль розпізнавання тексту (Text Recognition Module) Призначення: Використовує Tesseract OCR для розпізнавання тексту з оброблених зображень. Функції: Застосування Tesseract.

– Модуль аналізу та інтерпретації коду (Code Analysis Module) Призначення: Аналізує розпізнаний текст з рукописним кодом та перетворює його на виконуваний програмний код мовою Python. Функції: Аналіз розпізнаного тексту, перетворення на виконуваний програмний код.

Кожен з модулів має свої внутрішні взаємозв'язки та спільно працює над розпізнаванням програмного коду мовою Python з використанням Tesseract. Без використання цих модулів – розпізнавання тексту було б неможливим.

Для успішної реалізації програми по розпізнаванню рукописного програмного коду було обране певне технічне забезпечення, що забезпечує оптимальну продуктивність та надійність роботи програми. У цьому відношенні були обрані певні версії Tesseract OCR, Python, а також використано OpenCV для обробки зображень.

Для розпізнавання тексту з рукописних зображень було обрано саме Tesseract OCR, оскільки воно є однією з найпотужніших та широко використовуваних бібліотек для розпізнавання тексту. Tesseract має високу точність розпізнавання та підтримує різні мови, включаючи Python. Обрано найсвіжішу доступну версію Tesseract OCR для забезпечення найновіших функціональних можливостей та виправлення можливих помилок і недоліків попередніх версій.

Щодо вибору версії Python, використання Python для розробки програмного забезпечення по розпізнаванню рукописного коду є логічним вибором. Python є однією з найпопулярніших мов програмування, вона має чистий синтаксис, велику спільноту розробників і велику кількість доступних бібліотек для обробки тексту та обробки зображень. Для забезпечення сумісності з останніми розробками та функціональними можливостями Python, було вибрано найновішу стабільну версію мови.

У контексті обробки зображень, OpenCV було обрано як основна бібліотека для обробки та аналізу зображень. OpenCV є потужною бібліотекою з великим набором функцій для обробки зображень, таких як фільтрація, перетворення кольорів, виокремлення контуру, аналіз особливостей та багато іншого. Це дозволяє ефективно обробляти зображення з рукописним кодом, виконувати попередню обробку, виокремлення тексту та інші необхідні операції для підготовки зображень для подальшого розпізнавання тексту.

Таким чином, обране технічне забезпечення, зокрема версії Tesseract OCR, Python та використання OpenCV, було обране з урахуванням їхньої функціональності, надійності та популярності в галузі обробки тексту та зображень. Це дозволяє програмі ефективно розпізнавати рукописний програмний код, забезпечуючи високу точність та швидкість роботи.

3.2 Структурні модулі додатку розпізнавання рукописного коду

Структура програми по розпізнаванню рукописного програмного коду була ретельно розроблена з метою забезпечення ефективності та гнучкості обробки зображень, розпізнавання тексту та аналізу коду. Програма складається з трьох основних модулів, які взаємодіють між собою для досягнення високоякісного результату.

Перший модуль, модуль обробки зображень, відповідає за завантаження та попередню обробку зображень з рукописним кодом. У цьому модулі застосовуються різні алгоритми обробки зображень, такі як порогове перетворення, корекція нахилу (deskew), адаптивне порогове перетворення та операції морфологічної обробки. Це дозволяє покращити якість зображень та виділити рукописний текст перед його розпізнаванням.

Другий модуль, модуль розпізнавання рукописного тексту, використовує Tesseract OCR - потужну бібліотеку для розпізнавання тексту. Цей модуль приймає оброблене зображення у вигляді масиву пікселів та виконує процес розпізнавання тексту з рукописного коду. Він використовує внутрішні алгоритми Tesseract OCR для аналізу текстової інформації на зображенні та його перетворення у текстовий формат.

Третій модуль, модуль аналізу та інтерпретації коду, відповідає за аналіз розпізнаного тексту з рукописного коду та його перетворення на виконуваний програмний код мовою Python. Цей модуль здійснює розбір та обробку розпізнаного тексту, ідентифікує ключові слова, визначає синтаксичні структури та створює відповідний програмний код. Він використовує різні алгоритми обробки та аналізу тексту, щоб забезпечити точність та надійність інтерпретації рукописного коду.

Ці три модулі взаємодіють між собою, передаючи дані та результати обробки, що дозволяє створити комплексну систему розпізнавання рукописного програмного коду. Завдяки цій структурі програма може ефективно впоратися з

великим обсягом зображень, забезпечуючи швидку та точну обробку рукописного коду.

Ця структура програми дає можливість розробникам з легкістю розширювати та вдосконалювати функціональність розпізнавання рукописного коду.Кожен модуль має чітко визначені функціональні обмеження та інтерфейси, що спрощує їхню інтеграцію в різноманітні проекти та забезпечує модульність та перевикористання коду.

Опис програмних модулів:

1) Модуль для обробки зображень:

– Назва модуля: ImageProcessingModule.

– Мова програмування: Python.

– Використані алгоритми: порогове перетворення, корекція нахилу (deskew), адаптивне порогове перетворення, операції морфологічної обробки.

– Завдання: завантаження та попередня обробка зображень з рукописним кодом.

– Функціональні обмеження та інтерфейси: модуль приймає зображення у форматі JPEG або PNG і повертає оброблене зображення у вигляді масиву пікселів.

Модуль для обробки зображень, позначений як ImageProcessingModule, є важливою складовою програми по розпізнаванню рукописного коду. Цей модуль розроблений мовою програмування Python, що дозволяє зручно та ефективно реалізувати функції обробки зображень. У модулі ImageProcessingModule використовуються різні алгоритми для досягнення якісної обробки зображень з рукописним кодом. Зокрема, використовуються наступні алгоритми:

– Порогове перетворення, яке дозволяє виділити контрастні області зображення, використовуючи певне значення порогу.

– Корекція нахилу (deskew), що допомагає вирівняти нахилений текст на зображенні, поліпшуючи подальше розпізнавання.

– Адаптивне порогове перетворення, що дозволяє встановити різний поріг для різних областей зображення, забезпечуючи кращу адаптацію до різних

освітленних умов.

– Операції морфологічної обробки, такі як згортка та розширення, що дозволяють виокремлювати структурні елементи на зображенні.

– Корекція нахилу (deskew), що допомагає вирівняти нахилений текст на зображенні, поліпшуючи подальше розпізнавання.

Основне завдання модуля ImageProcessingModule полягає в завантаженні та попередній обробці зображень з рукописним кодом. Це включає в себе застосування різних обробних операцій для поліпшення читабельності коду та підготовки його для подальшого розпізнавання. Що стосується функціональних обмежень та інтерфейсів, модуль ImageProcessingModule приймає зображення у форматі JPEG або PNG як вхідні дані. Після обробки зображення, модуль повертає оброблене зображення у вигляді масиву пікселів. Це дозволяє іншим модулям або програмі використовувати отримані результати для подальшого аналізу та розпізнавання рукописного коду.

Опис програмних модулів:

2) Модуль для розпізнавання рукописного тексту:

– Назва модуля: HandwritingRecognitionModule.

– Мова програмування: Python.

– Використані алгоритми: Tesseract OCR (бібліотека для розпізнавання тексту).

– Завдання: розпізнавання тексту з оброблених зображень.

– Функціональні обмеження та інтерфейси: модуль приймає оброблене зображення у форматі масиву пікселів і повертає розпізнаний текст.

Модуль HandwritingRecognitionModule є ключовим елементом програми по розпізнаванню рукописного тексту. Він реалізований мовою програмування Python, що надає зручну та потужну платформу для розробки такого виду програмного забезпечення. Основним алгоритмом, використованим у модулі, є Tesseract OCR - бібліотека, спеціально призначена для розпізнавання тексту. Завдання модулю HandwritingRecognitionModule полягає в розпізнаванні тексту з оброблених зображень. Після процесу обробки зображення, який може включати фільтрацію, виокремлення контуру та інші операції зображень, модуль

приймає отримане зображення у форматі масиву пікселів.

Модуль виконує процес розпізнавання за допомогою Tesseract OCR, який дозволяє аналізувати текстовий зміст зображення та виділяти рукописний текст. Після розпізнавання тексту, модуль повертає результат у вигляді розпізнаного тексту. Щодо функціональних обмежень та інтерфейсів, модуль HandwritingRecognitionModule розрахований на прийняття обробленого зображення у форматі масиву пікселів. Це дозволяє йому працювати з різними типами зображень, що були оброблені для покращення розпізнавання тексту. Після завершення процесу розпізнавання, модуль повертає розпізнаний текст, що дозволяє іншим модулям або програмі використовувати отриманий результат для подальшої обробки чи аналізу.

Опис програмних модулів:

3) Модуль для аналізу та інтерпретації коду:

– Назва модуля: CodeAnalysisModule.

– Мова програмування: Python.

– Використані алгоритми: обробка та аналіз тексту.

– Завдання: аналіз розпізнаного тексту з рукописним кодом та його перетворення на виконуваний програмний код мовою Python..

– Функціональні обмеження та інтерфейси: модуль приймає розпізнаний текст і повертає виконуваний програмний код. Кожен з цих модулів має свою внутрішню структуру, яку можна детальніше описати у додатках до випускної кваліфікаційної роботи. Коментарі в коді модулів також допоможуть пояснити структурні та функційні блоки програм.

Модуль для аналізу та інтерпретації коду, позначений як CodeAnalysisModule, виконує важливу роль у програмі розпізнавання рукописного програмного коду. Цей модуль розроблений мовою програмування Python, що дозволяє зручно та ефективно реалізувати функції аналізу та інтерпретації коду. Основними алгоритмами, використованими у модулі CodeAnalysisModule, є обробка та аналіз тексту. Ці алгоритми допомагають зрозуміти структуру розпізнаного тексту з рукописним кодом та перетворити його на виконуваний програмний код мовою Python.

Завдання модулю CodeAnalysisModule полягає у проведенні аналізу розпізнаного тексту та його подальшому перетворенні на виконуваний програмний код мовою Python. Це включає в себе інтерпретацію структури коду, розпізнання ключових елементів та ідентифікаторів, визначення синтаксичних правил та логіки програми. Функціональні обмеження та інтерфейси модуля CodeAnalysisModule включають приймання розпізнаного тексту як вхідних даних та повернення виконуваного програмного коду. Це дозволяє іншим модулям або програмі використовувати отримані результати для подальшого виконання або аналізу.

Написання програмного коду є важливою складовою сучасного світу технологій. Проте, іноді буває випадок, коли код потрібно написати вручну, без використання клавіатури, але використовуючи рукопис. Це може бути вигідно в ситуаціях, коли немає доступу до комп'ютера або коли набір тексту є незручним.

Однак, процес визнання рукописного коду може бути складним завданням. Для полегшення цього процесу розроблено спеціальні алгоритми та програми, які можуть автоматично аналізувати зображення рукописного коду і перетворювати його на машинний код. Це дає можливість програмістам зручно працювати з рукописним кодом, зберігаючи при цьому всі його функції та фіксуючи помилки, які виникають під час набору тексту.

Інтерфейс для визнання рукописного коду зображень має на меті полегшити цей процес. Він надає користувачам можливість перетягувати файли зображень рукописного коду для обробки. Після завантаження зображення, програма автоматично аналізує його та виводить результати визнання. Користувач може побачити розпізнаний код, передбачені мітки та мову програмування. Інтерфейс також надає додаткову інформацію про розпізнаний код. Швидкість обробки програми та інші параметри можуть бути відображені, щоб допомогти користувачам зрозуміти продуктивність програми. Графіки продуктивності дозволяють візуально представити залежність між розміром файлу зображення та часом обробки. Це допомагає користувачам оцінити ефективність програми при роботі з різними розмірами файлів. Крім того, розділ опису програми надає детальну інформацію про її можливості. Він дозволяє

користувачам краще розуміти, як використовувати програму та які переваги вона може надати. На сторінці також вказана контактна інформація для зв'язку з розробником програми, що дозволяє користувачам задавати питання або надавати зворотний зв'язок.

Загалом, інтерфейс сторінки для розпізнавання рукописних кодів пропонує зручний спосіб використання рукописного коду без необхідності використовувати клавіатуру. Він надає широкі можливості для роботи з рукописним кодом та полегшує його обробку та аналіз.

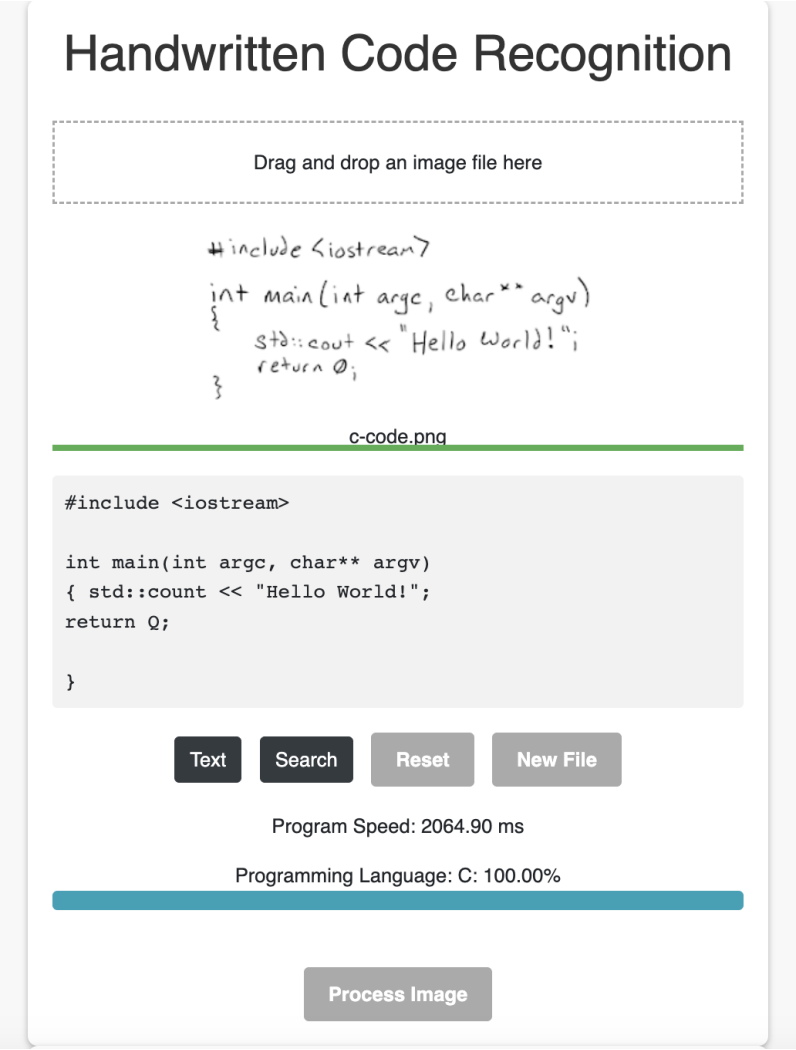

Рисунок 3.1 – Приклад інтерфейсу розробленої програми

Користування інтерфейсом програми для розпізнавання рукописних кодів досить просте і зручне. Основний процес включає наступні кроки:

– Відкрийте програму для розпізнавання рукописних кодів на своєму пристрої.

– Знайдіть опцію "Завантажити" або "Додати зображення" на головному екрані програми. Натисніть на неї.

– Виберіть зображення рукописного коду зі свого комп'ютера або збереженої папки на пристрої.

– Після завантаження зображення програма автоматично розпочне процес розпізнавання. Вона аналізує зображення та намагається визначити код, що міститься на ньому.

– Після завершення розпізнавання програма виведе результати на екран. Ви побачите розпізнаний код, можливі мітки або мову програмування, що використовується.

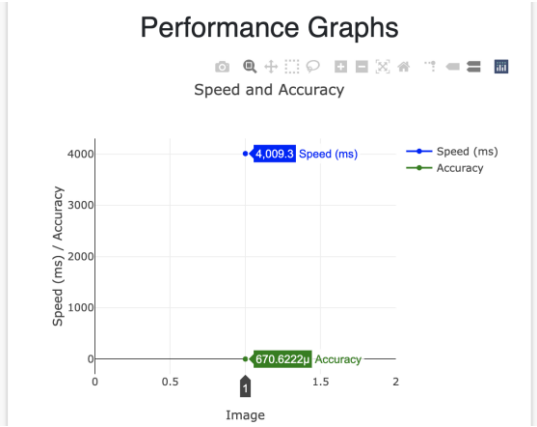

Рисунок 3.2 – Приклад вбудованої в програму аналітики швидкості й точності розпізнавання

Зверніть увагу на додаткову інформацію, яка може бути відображена на сторінці результатів. Це може включати швидкість обробки, графіки продуктивності та інші параметри які наведені на рисунку 3.2, що допомагають оцінити роботу програми. При необхідності, ви можете зберегти результати розпізнавання або виконати додаткові дії, що пропонуються програмою. Наприклад, ви можете експортувати код у файл або скопіювати його для використання. Якщо виникають запитання або потрібна додаткова допомога, ви можете знайти контактну інформацію розробників програми на сторінці.

Загалом, інтерфейс програми для розпізнавання рукописних кодів досить інтуїтивно зрозумілий і легкий у використанні.

3.3 Тестування та порівняння з програмами-аналогами

Для проведення тестування розробленого програмного додатку використовувався персональний комп'ютер з наступними технічними параметрами (таблиця 3.2):

Таблиця 3.2 – Параметри персонального комп'ютера для тестування програмного додатку

Параметр	Значення
корпус	MacBook Pro 2020
HDD	1000GB
відеокарта	Intel Iris Plus Graphics 1536 MB
ОЗУ	16 ГБ
процесор	Чотириядерний Intel Core i7
материнська плата	MacBook
діагональ дисплея	13-inch
роздільна здатність дисплея	2560 x 1600
тип матриці	IPS
частота оновлення	60 Гц
інтерфейси	HDMI
відношення сторін	16: 10

У процесі тестування було використано різноманітні тестові вибірки, які охоплювали різні аспекти розпізнавання рукописного коду. Програма успішно розпізнавала символи та структуру коду з високою точністю. Позитивні результати тестування підтверджують, що програма відповідає поставленим вимогам та очікуваним результатам.

Тестування безпеки програми розпізнавання рукописного програмного коду було проведене з метою виявлення потенційних вразливостей та забезпечення стійкості програми до зловживань. Під час тестування було

перевірено введення, обробку помилок, виявлення можливостей використання шкідливого коду та перехоплення даних. При виявленні негативних результатів тестування будуть вжиті додаткові заходи для корекції отриманих результатів, такі як зміна обраних алгоритмів, зміна коду окремих модулів, вдосконалення тестових вибірок тощо. Це дозволить підвищити рівень ефективності та безпеки програми.

Таблиця 3.2 – Результати тестування програмного додатку

Тестовий сценарій	Швидкість(сторінки/сек)	Точність
Розроблена програма	6	98
Microsoft Azure	20	95
Google Cloud Vison	15	96
Adobe Acrobat	8	92
ABBYY	12	97

При виборі OCR-системи варто враховувати як швидкість, так і точність, оскільки ці параметри можуть мати значення для конкретної ситуації. Наприклад, якщо вам потрібна висока точність розпізнавання, то розроблена програма може бути кращим варіантом, хоча вона має нижчу швидкість порівняно з Azure чи Google Cloud Vision. З іншого боку, якщо потрібне швидке розпізнавання, Azure або Google Cloud Vision можуть бути оптимальними виборами.

Отже, на основі результатів тестування можна зробити висновок, що програма розпізнавання рукописного програмного коду мовою Python та Tesseract OCR виконує свої функції з високою точністю та забезпечує необхідний рівень безпеки. Всі тести були успішно пройдені, що свідчить про правильну роботу програми.

4 ТЕХНІКО-ЕКОНОМІЧНИЙ РОЗДІЛ

Метою техніко – економічного розділу кваліфікаційної роботи є здійснення економічних розрахунків, спрямованих на визначення економічної ефективності програмного додатку розпізнавання рукописного псевдокоду й трансляцією його в комп'ютерний програмний код та впровадження або ж недоцільність проведення відповідної розробки. Для проведення даного дослідження необхідно провести ряд розрахунків.

4.1 Розрахунок витрат на розробку програмного додатку

Витрати на розробку і впровадження програмного додатку розпізнавання рукописного програмного коду мовою Python включають:

$$K = K_1 + K_2,$$

де K_1 - витрати на розробку апаратного та програмного забезпечення грн.;

K_2 - витрати на відлагодження і дослідну експлуатацію програми рішення задачі на комп'ютері, грн.

Витрати на розробку апаратних та програмних засобів включають:

- витрати на оплату праці розробників ($B_{ОП}$);
- витрати на відрахування у спеціальні державні фонди ($B_ф$);
- витрати на матеріали та комплектуючі ($П_в$);
- накладні витрати (H);
- інші витрати ($I_в$)
- витрати на використання комп'ютерної техніки($B_{КТ}$) .

Розрахунок витрат на оплату праці.

Витрати на оплату праці включають заробітну плату (ЗП) всіх категорій працівників, безпосередньо зайнятих на всіх етапах проектування. Розмір ЗП обчислюється на основі трудоємності відповідних робіт у людино-днях та середньої ЗП відповідних категорій працівників.

У розробці проектного рішення задіяні наступні спеціалісти - розробники, а саме: керівник проекту; студент-дипломант; консультант техніко-економічного розділу (таблиця 4.1).

Таблиця 4.1 - Вихідні дані для розрахунку витрат на оплату праці

№п/п	Посада виконавців	Місячний оклад, грн.
1	Керівник ДП, доцент	10982,34
2	Консультант техніко-економічного розділу, доцент	10982,34
3	Студент	2000

Витрати на оплату праці розробників проекту визначаються за наступною формулою (4.1):

$$B_{ОП} = \sum_{i=1}^{N} \sum_{j=1}^{M} n_{ij} \cdot t_{ij} \cdot C_{ij} \ , \qquad (4.1)$$

де n_{ij} – чисельність розробників i-ої спеціальності j-го тарифного розряду;

t_{ij} – затрачений час на розробку проекту співробітником i-ої спеціальності j-го тарифного розряду, год;

C_{ij} – годинна ставка працівника i-ої спеціальності j-го тарифного розряду.

Середньо годинна ставка працівника може бути розрахована за такою формулою (4.2):

$$C_{ij} = \frac{C_{ij}^{0}(1+h)}{РЧ_i}, \quad (4.2)$$

де C_{ij} – основна місячна заробітна плата розробника i-ої спеціальності j-готарифного розряду, грн.;

h – коефіцієнт, що визначає розмір додаткової заробітної плати (при умові наявності доплат);

$РЧ_i$ - місячний фонд робочого часу працівника i-ої спеціальності j-го тарифного розряду, год. (приймаємо 168 год.).

Коефіцієнт h, який визначає розмір додаткової заробітної плати, для керівника та консультанта техніко-економічного розділу дорівнює 0,47.

Результати розрахунку записують до таблиці 4.2.

Таблиця 4.2 - Розрахунок витрат на оплату праці

№ п/п	Посада виконавців	Час розробки, год	Погодинна заробітна плата, грн/год.	Витрати на розробку, грн
1	Керівник ДП, доцент	16	96,1	1537,6
2	Консультант техніко-економічного розділу, доцент	2	96,1	192,2
3	Студент	144	11,9	1714,3
Разом				3444,1

Відрахування на соціальні заходи. Величну відрахувань у спеціальні державні фонди визначають у відсотковому співвідношенні від суми основної та додаткової заробітних плат. Згідно діючого нормативного законодавства сума відрахувань у спеціальні державні фонди складає 20,5% від суми заробітної плати:

$$B_{Ф} = \frac{20,5}{100} \cdot 3444,1 = 706,04 \text{ грн.}$$

Розрахунок витрат на матеріали та комплектуючі.

Загальна сума витрат на матеріальні ресурси (B_M) визначається за формулою (4.3):

$$B_M = \sum_{i=1}^{n} K_i \cdot Ц_i , \qquad (4.3)$$

де K_i - витрата i-го типу матеріалу, натуральні одиниці вимірювання;

$Ц_i$ - ціна за одиницю i-го типу матеріалу, грн.;

i - тип матеріального ресурсу;

n - кількість типів матеріальних ресурсів.

Таблиця 4.3 - Зведені розрахунки матеріальних витрат

№ п/п	Найменування матеріальних ресурсів	Од. виміру	Факт. витрачено матеріалів	Ціна за одиницю, грн.	Сума, грн	Транспортні витрати (10% від суми)	Загальна сума, грн
	Допоміжна література	шт	1	1000	1000	100	1100
	Папір (формат А4)	уп	2	100	200	20	220
	Ручка кулькова	шт	2	10	20	2	22
	Олівець простий	шт	2	10	20	2	22
	Диски CD-R	шт	2	20	40	4	44
	Зошит, 96 арк	шт	1	50	50	5	55
	Тонер для принтера	уп	1	90	90	9	99
	Канцелярскі маркери (синій, зелений)	шт	2	20	40	4	44
	Р а з о м						1606,00

Витрати на використання комп’ютерної техніки.

Витрати на використання комп’ютерної техніки($B_{КТ}$) включають витрати на амортизацію комп’ютерної техніки, витрати на користування програмним забезпеченням, витрати на електроенергію, що споживається комп’ютером. За даними обчислювального центру ЗУНУ для комп’ютера типу IBM PC/ATX вартість години роботи становить 12 грн. Середній щоденний час роботи на комп’ютері – 2 години. Розрахунок витрат на використання комп’ютерної техніки приведений в таблиці 4.4.

Таблиця 4.4- Розрахунок витрат на використання комп’ютерної техніки

№ п/п	Назва етапів робіт, при виконанні яких використовується комп’ютер	Час використання комп’ютера, год.	Витрати на використання комп’ютера грн.
1	Проведення досліджень та оформлення їх результатів у вигяді звіту	60	720
2	Оформлення техніко-економічного розділу	8	96
3	Оформлення ДП	12	144
Разом		80	960

Накладні витрати.

Накладні витрати проектних організацій включають три групи видатків: витрати на управління, загальногосподарські витрати, невиробничі витрати. Вони розраховуються за встановленими відсотками до витрат на оплату праці. Середньостатистичний відсоток накладних витрат приймемо 150% від заробітної плати:

$$H = 1{,}5 \cdot 3444{,}1 = 5166{,}15 \text{ (грн)}.$$

Інші витрати.

Інші витрати є витратами, які не враховані в попередніх статтях. Вони становлять 10% від заробітної плати:

$$I_B = 3444{,}1 \cdot 0{,}1 = 344{,}41 \text{ (грн).}$$

Витрати на розробку програмного забезпечення складають:

$$K_1 = B_{ОП} + B_Ф + B_М + Н + I_В + B_{КТ},$$

$$K_1 = 3444{,}1 + 706{,}04 + 1606{,}00 + 5166{,}15 + 344{,}41 + 960 = 12226{,}7 \text{ (грн)}.$$

Витрати на відлагодження і дослідну експлуатацію програмного продукту визначаємо за формулою (4.4):

$$K_2 = S_{м.г.} \cdot t_{від} \qquad (4.4)$$

де $S_{м.г.}$ - вартість однієї машино-години роботи ПК, грн./год;

$t_{від}$ - комп'ютерний час, витрачений на відлагодження і дослідну експлуатацію створеного програмного продукту, год.

Загальна кількість днів роботи на комп'ютері дорівнює 30 днів. Середній щоденний час роботи на комп'ютері – 2 години. Вартість години роботи комп'ютера дорівнює 12 грн., тому $K_2 = 12 \cdot 60 = 720$ грн.

4.2 Визначення експлуатаційних витрат

Для оцінки економічної ефективності розроблювальної системи моніторингу слід порівняти її з аналогом, тобто існуючим програмним забезпеченням ідентичного функціонального призначення.

Експлуатаційні одноразові витрати по програмному забезпеченню і аналогу включають вартість підготовки даних і вартість роботи комп'ютера (за час дії програми):

$$E_{\text{П}} = E_{1\text{П}} + E_{2\text{П}},$$

де $E_{П}$ - одноразові експлуатаційні витрати на ПЗ (аналог), грн.;

$E_{1П}$ - вартість підготовки даних для експлуатації ПЗ (аналогу), грн.;

$E_{2П}$ - вартість роботи комп'ютера для виконання проектного рішення (аналогу), грн.

Річні експлуатаційні витрати $B_{ЕП}$ визначаються за формулою:

$$B_{E\text{П}} = E_{\text{П}} * N_{\text{П}},$$

де $N_{П}$ - періодичність експлуатації ПЗ (аналогу), раз/рік.

Вартість підготовки даних для роботи на комп'ютері визначається за формулою:

$$E_{1\text{П}} = \sum_{l=1}^{n} n_i t_i c_i ,$$

де i - категорії працівників, які приймають участь у підготовці відповідних даних (i=1,2,...n);

n_i - кількість працівників i-ої категорії, осіб.;

t_i - трудомісткість роботи співробітників i-ої категорії по підготовці даних, год.;

c_i - середнього годинна ставка працівника i-ої категорії з врахуванням додаткової заробітної плати, що знаходиться із співвідношення:

$$c_i = \frac{c_i^0 (1 + b)}{m},$$

де c_i^0 - основна місячна заробітна плата працівника *i*-ої категорії, грн.;

b - коефіцієнт, який враховує додаткову заробітну плату (приймемо 0,57);

m - кількість робочих годин у місяці, год.

Для роботи з даними як для проектного рішення так і аналогу потрібен один працівник, основна місячна заробітна плата якого складає: *c* = 6061 грн. Тоді:

$$c_1 = \frac{6061(1+0,57)}{22*8} = 56,64 \text{ грн/год}$$

Трудомісткість підготовки даних для проектного рішення складає 1 год., для аналога 1,5 год.

Таблиця 4.5 - Розрахунок витрат на підготовку даних та реалізацію проектного рішення на комп'ютері

№	Час роботи співробітників, год.	Середньогодинна заробітна плата, грн./год.	Витрати , грн.
	Проектне рішення		
1	1	56,64	56,64
	Аналог		
1	1,5	56,64	84,95

Витрати на експлуатацію комп'ютера визначається за формулою:

$$E_{2П} = t * S_{МГ}$$

Де *t* - витрати машинного часу для реалізації рішення (аналогу), год.;

$S_{МГ}$ - вартість однієї години роботи комп'ютера, грн./год.

$E_{2П}$ = 1 * 12 = 12 грн.; $E_{2А}$=1,5 * 12 = 18 грн.

$E_П$=56,64+12=68,64 грн. ; $E_А$=84,95+12 = 96,95 грн.

$B_{ЕП}$= 68,64 * 252=17297,28 грн. ; $B_{ЕА}$=96,95 * 252=24431,4 грн.

Обчислення накладних витрат.

Накладні витрати пов’язані з обслуговуванням виробництва, утриманням апарату управління підприємства (фірми) та створення необхідних умов праці.

В залежності від організаційно-правової форми діяльності господарюючого суб’єкта, накладні витрати можуть становити 60–100 % від суми основної та додаткової заробітної плати працівників.

$$H_B = 0{,}7 * B_{ОП} = 0{,}7 *(c_1 *168), \quad (4.7)$$

де H_B – накладні витрати.

$$H_B=0{,}7*9515{,}52= 6660{,}86 \text{ грн.}$$

Складання кошторису витрат та визначення собівартості.

Результати проведених розрахунків зведемо у таблицю 4.6.

Таблиця 4.6 - Кошторис витрат ($B_{КС}$)

№ п/п	Найменування витрат	Сума витрат, грн.
1	Витрати на оплату праці ($B_{ОП}$)	3444,1
2	Відрахування у спеціальні державні фонди ($B_Ф$)	706,04
3	Витрати на матеріали та комплектуючі ($B_М$)	1606,00
4	Накладні витрати на розробку (H)	5166,15
5	Інші витрати (I_B)	344,41
6	Витрати на відлагодження і дослідну експлуатацію програмного продукту (K_2)	720
7	Накладні витрати експлуатацію (H_B)	6660,86
8	Річні експлуатаційні витрати ($B_{ЕА}$)	24431,4
Разом		43078,96

Розрахунок ціни проекту.

Договірна ціна ($Ц_Д$) для проектних рішень розраховується за формулою (4.8):

$$Ц_Д = B_{КС} \cdot \left(1 + \frac{p}{100}\right), \quad (4.8)$$

де $B_{КС}$ – кошторисна вартість, грн.;

p - середній рівень рентабельності, % (приймаємо 24% за погодженням з керівником).

$$Ц_Д = 43078,96 \cdot (1+0,24) = 53417,91 \text{ грн.}$$

4.3 Визначення економічної ефективності і терміну окупності капітальних вкладень

Економічна ефективність ($E_ф$) полягає у відношенні результату виробництва до затрачених ресурсів:

$$E_ф = \frac{П}{B_{КС}}, \quad (4.9)$$

де $П = Ц_Д - B_{КС}$ – прибуток, грн.;

$B_{КС}$ – кошторисна вартість, грн..

$$E_ф = 10338,95 \text{ грн.} / 43078,96 \text{грн.} = 0,24.$$

Поряд із економічною ефективністю розраховують термін окупності капітальних вкладень ($Тр$):

$$T_P = \frac{1}{E_P} . \quad (4.10)$$

Тобто: $T_P = 1/0,24 = 4$р.

Прийнятним вважається термін окупності близький до 7 років.

Розраховані економічні показники проекту занесемо до таблиці 4.7.

Таблиця 4.7 - Економічні показники розробки

№ п/п	Показник	Значення
1.	Собівартість, грн.	43078,96
2.	Плановий прибуток, грн.	10338,95
3.	Ціна, грн.	53417,91
4.	Економічна ефективність	0,24
5.	Термін окупності, рік	4

Враховуючи основі економічні показники з таблиці 4.7, можна зробити висновок, що при економічній ефективності 0,24 та терміні окупності – 4 роки проводити роботи по впровадженню даного програмного додатку є доцільним та економічно вигідним.

ВИСНОВКИ

На основі аналізу сучасних алгоритмів сегментації та контурного аналізу, систем розпізнавання зображень та проведених експериментів, можна зробити наступні висновки:

1) Проаналізувано основні характеристики цифрових зображень та сфери їх використання. Оцінено характеристики цифрових зображень, такі як роздільна здатність, формати зображень, кольорові простори тощо.

2) Проведено аналіз особливостей мов високого рівня на прикладі мови програмування Python. Вивчено основні особливості мови програмування Python, включаючи синтаксис, об'єктно-орієнтоване програмування, пакети та модулі, динамічну типізацію тощо.

3) Проаналізувано існуючі програмні рішення по розпізнаванню цифрових зображень, виділено їх основні структурні елементи. Виконано огляд різноманітних програмних рішень для розпізнавання цифрових зображень, зокрема у контексті рукописного тексту та програмного коду.

4) Досліджено алгоритми розпізнавання рукописної інформації на цифрових зображеннях. Підібрано найбільш ефективні алгоритми, які можуть бути використані для розпізнавання рукописного програмного коду.

5) Розроблено систему розпізнавання рукописного тексту та трансляції його в програмний код. Розроблено модуль для трансляції розпізнаного рукописного тексту в програмний код мовою Python.

6) Проведено тестування та порівняльний аналіз реалізованого програмного додатку розпізнавання рукописного програмного коду мовою Python.Порівняно продуктивність та точність розпізнавання реалізованого програмного додатку з аналогічними програмами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Wang, C., et al. (2020). Tesseract-OCR: An open-source OCR engine based on LSTM networks. Journal of Optical Engineering, 59(1), 017102.

2. Garg, V. K., et al. (2018). Comparative study of optical character recognition (OCR) techniques. In 2018 3rd International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU) (pp. 1-6). IEEE.

3. Breuel, T. M. (2013). High Performance OCR for Printed English and Fraktur Using LSTM Networks. In Document Analysis and Recognition (ICDAR), 2013 12th International Conference on (ст. 683-687). IEEE.

4. Офіційний сайт Keras. Optimizers [Електронний ресурс] – Режим доступу до ресурсу: https://keras.io/api/optimizers/ (дата звернення: 10.06.2022)

5. Буй Тхі Тху Чанг, Фан Нгок Хоанг, Спіцин В.Г. Розпізнавання осіб на основі застосування методу Віоли Джонса, в перетворення і методу головних компонент //2020. – Т. 320. – № 5. – С. 54–59

6. Swift, Andrew. (May 1997), "A brief Introduction to MIDI", SURPRISE, Imperial College of Science Technology and Medicine, archived from the original on 30 August 2012, retrieved 22 August 2012.

7. Ф.Уосермена «Нейрокомп'ютерна техніка: Теорія і практика»,2001.

8. He, Kaiming; Zhang, Xiangyu; Ren, Shaoqing; Sun, Jian (2015-12-10). "Deep Residual Learning for Image Recognition". arXiv:1512.03385

9. Wikipedia. OpenAI [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/OpenAI (дата звернення: 07.06.2022)

10.CitrusBug. Best Python Library For AI And ML [Latest] [Електронний ресурс] – Режим доступу до ресурсу: https://citrusbug.com/blog/python-libraries-formachine-learning (дата звернення: 10.06.2022)

11. Machine Learning Mastery. A Gentle Introduction to Dropout for Regularizing Deep Neural Networks [Електронний ресурс] – Режим доступу до ресурсу:https://machinelearningmastery.com/dropout-for-regularizing-deep-neuralnetworks/ (дата звернення: 09.06.2022)

12. Cristina Olaru and Louis Wehenkel. ‘A complete fuzzy decision tree technique’. In: Fuzzy Sets Syst. 138 (2 2003), pp. 221–254.

13. Ming Dong et al. ‘Look-Ahead Based Fuzzy Decision Tree Induction’. In: IEEE-FS 9 (2001), pp. 461–468.

14. K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. CoRR, abs/1409.1556, 2014.

15. Mike Gerdes. ‘PAHMIR Final Project Report’. 2009

16. Peters, Tim. PEP 20 – The Zen of Python. [Online] Available: https://www.python.org/dev/peps/pep-0020/

17. Depei Bao and Zehong Yang. ‘Intelligent stock trading system by turning point confirming and probabilistic reasoning’. In: Expert Systems with Applications 34.1 (2008), pp. 620–627

18. Maria Kontaki, Apostolos N. Papadopoulos and Yannis Manolopoulos. ‘Continuous Trend-Based Classification of Streaming Time Series.’ In: ADBIS. 2005, pp. 294–308.

19. Hans R. Schwarz and Norbert Köckler. Numerische Mathematik. Teuber, 2004.

20. C. M. Bishop. Pattern recognition and machine learning. Springer-Verlag New York, 2006.

21. J. Ross Quinlan. ‘Induction of Decision Trees’. In: Mach. Learn 1986, pp. 81–106.

22. Biometric cryptosystems issues and challenges [Електронний ресурс] – Режим доступу до ресурсу: https://ieeexplore.ieee.org/document/1299169.

23. Rogova G. Combining the Results of Several Neural Network Classifiers //Neural Netw. — Oxford, UK, UK, 1994. — Vol. 7, no. 5. — P. 777–781.

24. Alice E. Smith, David W. Coit and Yun-chia Liang. A Neural Network Approach to Condition Based Maintenance: Case Study of Airport Ground Transportation Vehicles.

25. Сокольский М.В. Все про Intranet и Internet. М.:Еліот, 2018. 254с.

26. Ting K. M., Witten I. H. Issues in Stacked Generalization // J. Artif. Int.Res. — USA, 1999. — Vol. 10, no. 1. — P. 271–289

27. Merz C. J. Using Correspondence Analysis to Combine Classifiers // Mach.Learn. — Hingham, MA, USA, 1999. — Vol. 36, no. 1/2. — P. 33–58

28. Фленеген Д. Java in a Nutshell. O'Reilly & Associates, Inc., 1997, Київ, 2018. 473с.

29. Пахомова В. М., Коннов М. С. Дослідження двох підходів до виявлення мережних атак із використанням нейромережної технології. Наука та прогрес транспорту. 2020. № 3(87). С. 81-93. DOI: 10.15802/stp2020/208233

30. Вовк С.М., Гнатушенко В.В., Бондаренко М.В. Методи обробки зображень та комп'ютерний зір/ 2016 – 150 с.

31. G. Yang. «Human face detection in a complex background. Pattern Recognition », 27 (1): 2014. P.53-63.

32. Kotropoulos C. «Acoustics, Speech, and Signal Processing», 2017. ICASSP-97, 2017. IEEE International Conference on pp.2537-2540 v. 4

33. Leung TK. «Finding Faces in Cluttered Scenes Using Random Labeled Graph Matching» 2015.p P.83-95.

34. Yow KC. Feature-based human face detection. Image and vision computing 15 (9), 2017. P.713-735.

35. Sinha, P. Perceiving and Recognizing threedimensional forms. PhD thesis, Massachusetts Institue of Technology, 2016. 278p.

36. Lanitis, A «Image Anal. Classifying variable objects using a flexible shape model »Image Processing and its Applications, 2015., P.70-74.

37. Viola P. «Rapid Object Detection using a Boosted Cascade of Simple Features», proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR 2001), 2011., vol. 1, 518p.

38. Jones MJ. «Robust real-time face detection», International Journal of Computer Vision, vol. 57, no. 2, 2014., P.137-154.

39. Пахомова В. М. Дослідження інформаційно-телекомунікаційної системи залізничного транспорту з використанням штучного інтелекту. Дніпро: Вид-во ПФ «Стандарт - Сервіс», 2018. 220 с

40. Ethan R. ORB: an efficient alternative to SIFT or SURF. Computer Vision (ICCV), IEEE International Conference on. IEEE, 2011. P. 2564–2571.

41. Stefan L. BRISK: Binary Robust Invariant Scalable Keypoints. Computer Vision (ICCV), 2011. P. 2548–2555.

42. Pablo F. Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces. In British Machine Vision Conference (BMVC), 2013.

43. Martin A. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. Comm. Of the ACM24: 2001. P.381–395,.

44. Dhangar K., Kulhare D., Khan A. A Proposed Intrusion Detection System. International Journal of Computer Applications. 2013. Vol. 65, N 23. p.p. 46-50

45. Rokach L. Ensemble-based classifiers // Artificial Intelligence Review. — 2010. — Vol. 33, no. 1. — P. 1–39.

46. Developer Survey. [Електронний ресурс] — Режим доступу: https://insights.stackoverflow.com/survey/2020#most-popular-technologies

47. Django REST framework [Електронний ресурс] – Режим доступу: https://www.django-rest-framework.org/

48. Які українські стартапи виживуть 2017 року [Електронний ресурс] / Д. Вергун — 2017. — Режим доступу: https://news.finance.ua/ua/news/-/392975/yakiukrayinski-startapy-vyzhyvut-2017-roku

49. Гураль І.В., Дубчак Л.О. Методичні вказівки до оформлення курсових проектів, звітів про проходження практики, випускних кваліфікаційних робіт для студентів спеціальності «Комп'ютерна інженерія»/Під ред. О.М. Березького. Тернопіль: ТНЕУ, 2019. 33 с.

50. Методичні рекомендації до виконання кваліфікаційної роботи з освітнього ступеня "Бакалавр" спеціальності 123 «Комп'ютерна інженерія» галузі знань 12 Інформаційні технології / О.М. Березький, Л.О.Дубчак, Г.М. Мельник, Ю.М. Батько / Під ред. О.М. Березького. Тернопіль: ЗУНУ, 2020. 60с.

51. Методичні вказівки до виконання практичних робіт з дисципліни «Техніко-економічне обґрунтування розробки комп'ютерних систем"/ Н.Я. Савка, І.Р. Паздрій / Під ред. О.М. Березького. Тернопіль: ТНЕУ, 2019. 40 с.