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

Костенюк Анатолій Васильович

Алгоритми реконструкції шрифтів при оптичному розпізнаванні тексту / Font reconstruction algorithms for optical text recognition

спеціальність: 123 - Комп'ютерна інженерія
освітньо-професійна програма - Комп'ютерна інженерія
Кваліфікаційна робота

Виконав студент групи КІм-21
А.В. Костенюк
_____

Науковий керівник:
к.т.н., Г.М. Мельник
_____

Кваліфікаційну роботу допущено
до захисту:

"___" _____ 20___ р.

Завідувач кафедри
_____Л. О. Дубчак

Тернопіль – 2023

РЕЗЮМЕ

Кваліфікаційна робота на тему «Алгоритми реконструкції шрифтів при оптичному розпізнаванні тексту» зі спеціальності 123 «Комп'ютерна інженерія» освітнього ступеня «магістр» написана обсягом 80 сторінок і містить 9 ілюстрацій, 6 таблиць, 3 додатки та 50 джерел за переліком посилань.

Метою кваліфікаційної роботи є розроблення алгоритму реконструкції шрифтів при оптичному розпізнаванні тексту.

Методи дослідження: методи системного аналізу, математичного моделювання.

Наукова новизна одержаних результатів. Розроблено алгоритми реконструкції шрифтів на основі контурних ознак та співставлення шаблонів, що дозволило підвищити точність розпізнавання тексту.

Практичне значення отриманих результатів. Розроблені алгоритми дозволяють розпізнавати текст із значними спотвореннями символів. Отримані результати опубліковані в межах VIII Науково-практичної конференції молодих вчених і студентів «Інтелектуальні комп'ютерні системи та мережі».

Розроблено алгоритми реконструкції шрифтів на основі обчислення топологічних характеристик ключових точок скелетного представлення контуру, що дозволило з'єднувати фрагменти символу шрифту. Розроблено алгоритм виявлення символів на основі проекції, що дозволило виявляти пробіли між символами шрифту в стрічці тексту. Розроблено алгоритм реконструкції шрифту на основі шаблонів, що дозволило відновлювати невпевнено розпізнані символи і підняти точність розпізнавання. Результати показали що в залежності від рівня спотворень підвищення точності розпізнавання складає від 25% до 5%.

КЛЮЧОВІ СЛОВА: ШРИФТ, ОПТИЧНЕ РОЗПІЗНАВАННЯ ТЕКСТУ, КОНТУР, ТЕКСУТРА.

RESUME

Qualification work on «Font reconstruction algorithms for optical text recognition» in the specialty 123 "Computer Engineering" educational degree "Master" is written in 80 pages and contains 7 illustrations, 6 tables, 3 appendices and 50 sources by reference.

The purpose of the qualification work is to develop an algorithm for font reconstruction in optical text recognition.

Research methods: methods of system analysis, mathematical modeling.

Scientific novelty of the results. The algorithms for font reconstruction based on contour features and pattern matching have been developed, which has improved the accuracy of text recognition.

Practical significance of the results. The developed algorithms allow recognizing text with significant character distortions. The results were published within the framework of the VIII Scientific and Practical Conference of Young Scientists and Students "Intelligent Computer Systems and Networks".

Algorithms for font reconstruction based on the calculation of topological characteristics of key points of the skeletal representation of the contour were developed, which allowed to connect fragments of a font character. A character detection algorithm based on projection was developed, which made it possible to detect gaps between font characters in a text string. A template-based font reconstruction algorithm was developed, which allowed restoring uncertainly recognized characters and improving recognition accuracy. The results show that, depending on the level of distortion, the increase in recognition accuracy ranges from 25% to 5%.

KEYWORDS: FONT, OPTICAL TEXT RECOGNITION, CONTOUR, TEXTURE.

ЗМІСТ

ВСТУП

Актуальність. На сьогоднішній день задача оптичного розпізнавання текстів (ОРТ) виступає як ключовий інструмент у процесах цифровізації, автоматизації та аналітики. Отже, впровадження ОРТ технологій дозволяє значно спростити і прискорити доступ до великих обсягів текстової інформації, яка зберігається у нецифрових форматах.

Оптичне розпізнавання текстів знаходить своє застосування у різноманітних сферах: від сканування та конвертації документів у банківській справі до аналізу історичних записів для наукових досліджень, від обробки медичних форм до вдосконалення систем розпізнавання номерних знаків автомобілів. Кожна з цих областей вимагає високої точності перетворення зображення тексту в електронний формат, щоб далі можна було з легкістю проводити пошук, редагування, зберігання та обмін інформацією.

Проте, процес оптичного розпізнавання текстів не позбавлений проблем. Серед них — спотворення та помилки, які можуть значно вплинути на кінцевий результат розпізнавання. Якість зображення, на якому представлено текст, може бути погіршена через багато факторів: низьку роздільну здатність, шуми, неправильне освітлення, спотворення через згини або вологу та ін. Також, важливим фактором є стан самого тексту — розмитість друку, незвичайні шрифти або рукописний текст, який особливо складно інтерпретувати. Ці умови можуть призвести до неправильного розпізнавання символів, словосполучень, а іноді і цілих речень.

Важливість розроблення нових алгоритмів покращення точності OCR неможливо переоцінити. Сучасні дослідження у цій сфері зосереджені на створенні адаптивних і гнучких систем, здатних ефективно працювати з різними типами текстів і під різними кутами зору. Новітні розробки використовують машинне навчання та штучний інтелект, щоб поліпшити здатність алгоритмів "розуміти" структуру і семантику тексту, що, у свою чергу, зменшує кількість помилок та спотворень.

Розвиток ОРТ технологій відкриває нові перспективи для архівування документів, доступу до забутого культурного спадку, створення інтерактивних навчальних платформ та розвитку ефективних систем управління документообігом. З точки зору бізнесу, поліпшення OCR може призвести до зменшення витрат, збільшення продуктивності та покращення аналітики даних. У світі, де обсяги інформації зростають експоненціально, здатність швидко та точно перетворювати фізичні дані у цифровий формат є невід'ємною частиною цифрової трансформації.

Метою кваліфікаційної роботи є розроблення алгоритму реконструкції шрифтів при оптичному розпізнаванні тексту.

Об'єкт дослідження – методи оптичного розпізнавання тексту.

Предмет дослідження – алгоритми підвищення точності оптичного розпізнавання тексту.

Для досягнення поставленої мети роботи можна успішно досягти шляхом виконання таких завдань:

- провести аналітичний огляд існуючих алгоритмів аналізу шрифтів.
- розробити алгоритм реконструкції шрифтів;
- розробити алгоритм виявлення символів;
- розробити алгоритм розпізнавання символів шрифту;
- розробити і реалізувати програмне забезпечення оптичного розпізнавання тексту.

Наукова новизна одержаних результатів. Розроблено алгоритми реконструкції шрифтів на основі контурних ознак та співставлення шаблонів, що дозволило підвищити точність розпізнавання тексту.

Практичне значення отриманих результатів. Розроблені алгоритми дозволяють розпізнавати текст із значними спотвореннями символів.

Апробація роботи. Отримані результати опубліковані в межах VIII Науково-практичної конференції молодих вчених і студентів «Інтелектуальні комп’ютерні системи та мережі», опубліковано тези доповіді [1,2].

Впровадження результатів ДР. Результати роботи рекомендовано до впровадження на підприємстві (додаток Б).

Кваліфікаційна робота складена із трьох розділів, висновків, списку використаних джерел та додатків.

Перший розділ, присвячений огляду алгоритмів розпізнавання текстових документів, проблем розпізнавання спотворених зображень, існуючих програмних засобів оптичного розпізнавання тексту.

Другий розділ присвячено розробці алгоритмів реконструкції шрифтів на основі контурів шрифту та співставлення шаблонів.

Третій розділ присвячений розробці програмного забезпечення для оптичного розпізнавання тексту. Проведено експериментальне дослідження точності розпізнавання.

1 ОГЛЯД АЛГОРИТМІВ РОЗПІЗНАВАННЯ ТЕКСТОВИХ ДОКУМЕНТІВ

1.1 Задачі машинного зору та їх різноманітність

Сьогодні чітко визначена розмежувальна лінія між монокулярним і бінокулярним комп'ютерним зором. Перший варіант включає в себе вивчення та розробки, які зосереджені на обробці інформації, отриманої з однієї камери або окремо від кожної камери. Другий варіант включає в себе дослідження та розробки, які стосуються обробки інформації, одночасно отриманої від двох або більше камер. В таких системах використовується кілька камер для вимірювання глибини спостереження і вони відомі як стереосистеми.

В наші дні, теорія комп'ютерного зору стала повноцінним розділом кібернетики, що базується на наукових та практичних знаннях. Щорічно видається сотні книг і монографій, проводиться безліч конференцій і симпозіумів, розробляється різноманітне програмне і апаратно-програмне забезпечення. Існують науково-суспільні організації, які підтримують та просувають дослідження в сфері сучасних технологій, включаючи технології комп'ютерного зору.

В загальному розумінні, завдання систем машинного зору включає в себе отримання цифрового зображення, обробку зображення для виділення важливої інформації на ньому та математичний аналіз отриманих даних для вирішення поставлених завдань. Проте машинний зір дозволяє розв'язувати різноманітні завдання, які можна умовно поділити на чотири групи: (рисунок 1.1) [7]:

Задача розпізнавання положення.

Метою машинного зору в цьому випадку є визначення просторового розташування об'єкта відносно зовнішньої системи координат або статичного положення об'єкта відносно системи координат з початком відліку в самому об'єкті. Ця інформація передається системі керування або контролеру.

Наприклад, вантажно-розвантажувальний робот може використовувати машинний зір для ефективного переміщення об'єктів різної форми з одного місця в інше. Інтелектуальна обробка машинного зору включає в себе визначення оптимальної системи координат та її центру для точного локалізування центру

ваги деталі, що дозволяє роботу забезпечити правильне захоплення та переміщення деталі в необхідне місце.

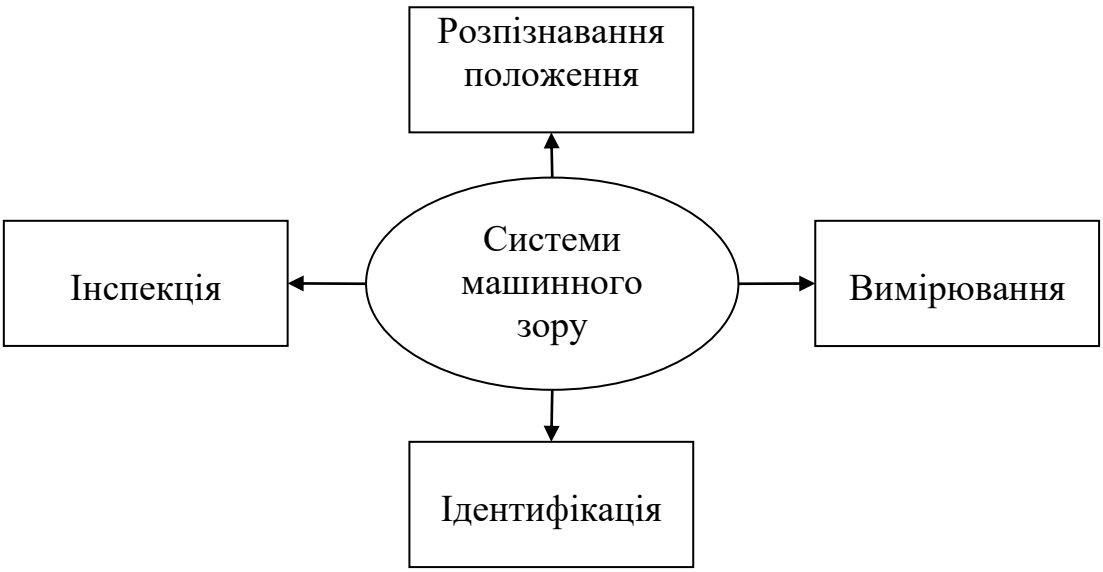


Рисунок 1.1 - Задачі машинного зору

Задача вимірювання. В контексті цих застосувань, головною метою відеокамери є точне вимірювання різних фізичних параметрів об'єкта. Ці фізичні параметри можуть включати лінійний розмір, діаметр, кривизну, площу, висоту та кількість. Наприклад, цю задачу можна реалізувати через вимірювання різних діаметрів горлечка скляної пляшки.

Інспекція. У застосуваннях, пов'язаних з інспекцією, завдання машинного зору полягає у підтвердженні певних властивостей об'єкта, таких як наявність або відсутність етикетки на пляшці, наявність болтів для проведення операції складання, наявність шоколадних цукерок у коробці або виявлення різних дефектів.

Ідентифікація. У завданнях ідентифікації, головним призначенням відеокамери є зчитування різних кодів (наприклад, штрих-кодів, 2D-кодів і інших) з метою їх розпізнавання камерою або системним контролером. Крім того, до цієї групи завдань можна віднести системи безпеки, такі як ідентифікація особистості та транспортних засобів, а також системи виявлення руху.

З огляду на різноманітність завдань, які вирішує машинний зір, можна виділити широкий спектр областей застосування цієї технології. Проте слід зауважити, що сучасні можливості систем машинного зору поки що обмежені, і структура попиту на ринку формується враховуючи ці обмеження.

Нижче представлена структура ринкового попиту в контексті проектних тематик (див. рисунок 1.2) [4].

Рисунок 1.2 - Структура ринку

З графіка видно наступне.

1. Приблизно 50% всіх систем машинного зору застосовуються в завданнях контролю якості, що включає інспекційні завдання машинного зору. Основна функція цих систем - візуальний нагляд за процесом збирання, коліром і якістю поверхні продукції, зовнішнім виглядом і станом упаковки, правильністю і читабельністю етикеток, а також рівнем рідини у різних видів тари і т.д. Приблизно 10% таких завдань виконуються системами тривимірного зору. Окремою областю застосування систем машинного зору на виробництві є вимірювання різних параметрів технологічних процесів, включаючи розміри об'єктів.

2. Приблизно 20% попиту припадає на системи машинного зору для проектів автоматизації виробництва і впровадження промислових роботів. Ці системи спрощують виконання різноманітних високоточних операцій, таких як

збирання і розбирання, фасування, фарбування, зварювання та утилізація, полегшують транспортування вантажів і застосовуються в системах обліку, маркування, реєстрації і сортування продукції. Крім того, вони використовуються для вирішення інспекційних завдань і завдань локалізації для ефективного управління роботами.

3. Близько 17% всіх продажів систем машинного зору припадають на широко відомі і ефективні системи OCR/OCV для розпізнавання друкованих символів і штрих-кодів. Ці системи вирішують завдання ідентифікації.

4. Ринок систем машинного зору для не-виробничих (розважальних, побутових, наукових) роботів становить приблизно 13% від загального обсягу ринку.

У системах машинного зору використовуються різноманітні технології і методи для вирішення вищезазначених завдань. Ось основні методи обробки зображень:

1. Лічильник пікселів: Цей метод обчислює кількість світлих і темних пікселів на зображенні і на основі цього робить висновки про зображення.

2. Виділення зв'язаних областей: Зв'язана область на зображенні - це самостійна семантична одиниця, що дозволяє проводити подальший геометричний, логічний, топологічний та інший аналіз зображення. Використовується для ідентифікації об'єктів.

3. Бінаризація: Цей метод перетворює зображення у сірих відтінках на бінарне (білі та чорні пікселі), що полегшує аналіз.

4. Гістограма і гістограмна обробка: Гістограма вказує на частоту зустрічі пікселів однакової яскравості, що корисно для аналізу розподілу яскравості на зображенні.

5. Сегментація: Цей метод допомагає знайти та підрахувати деталі на зображенні, розбиваючи його на несхожі за певними ознаками області.

6. Читання штрих-кодів: Системи, що вирішують завдання декодування 1D і 2D кодів для зчитування і сканування.

7. Оптичне розпізнавання символів: Автоматизоване читання тексту, такого як серійні номери або інші символи.

8. Вимірювання: Вимір розмірів об'єктів у дюймах або міліметрах.

9. Зіставлення шаблонів: Пошук, підбір і визначення конкретних моделей.

10. Алгоритми ідентифікації особистості за райдужною оболонкою ока.

11. Різні методи відновлення форми об'єктів на основі зображень.

Ці різноманітні методи і технології дозволяють вирішувати широкий спектр завдань у системах машинного зору.

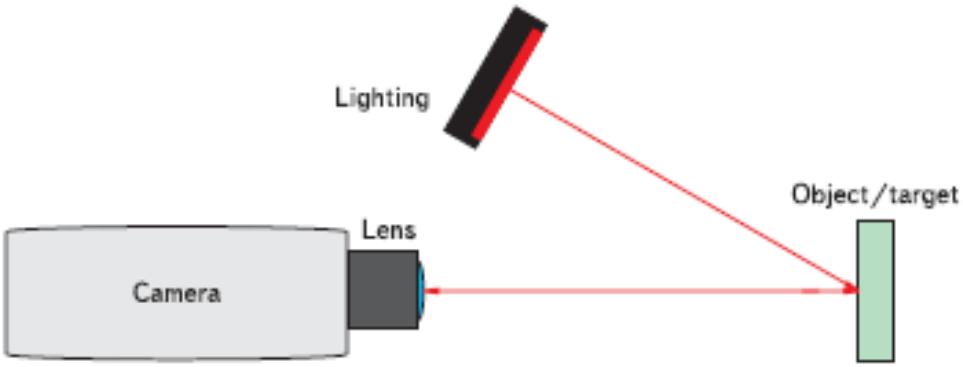


Рисунок 1.3 - Компоненти зразкової системи машинного зору.

Об'єктив камери відіграє ключову роль, дозволяючи фокусуватись на об'єктах на певній відстані для отримання ясних зображень. Якщо об'єкт знаходиться за межами цієї відстані, зображення стає розмитим, що ускладнює обробку відео. В машинному зорі переважно використовуються об'єктиви з фіксованою фокусною відстанню або з можливістю ручного налаштування фокусу, на відміну від звичайних фотоапаратів з автофокусом. Різноманітність об'єктивів (стандартні, телеоб'єктиви, з широким кутом огляду, макрооб'єктиви тощо) дає змогу вирішити специфічні завдання, тому правильний вибір оптики є критичним для проектування системи машинного зору.

Освітлення є ще одним важливим елементом. Використовуючи різні типи освітлення, можна розширити спектр завдань, які може вирішувати система машинного зору. Серед різноманіття варіантів освітлення, світлодіодне освітлення є найбільш затребуваним через його високу яскравість, тривалий термін служби та низьке споживання енергії.

Послідовність процесів у системі машинного зору можна зобразити на схемі (рисунок 1.4). Зображення, захоплене камерою, передається до захоплювача кадрів або зберігається у пам'яті комп'ютера. Захоплювач кадрів перетворює дані з камери у цифровий формат (зазвичай у вигляді двовимірного масиву чисел) і розміщує зображення в пам'яті комп'ютера для подальшої обробки програмним забезпеченням машинного зору.

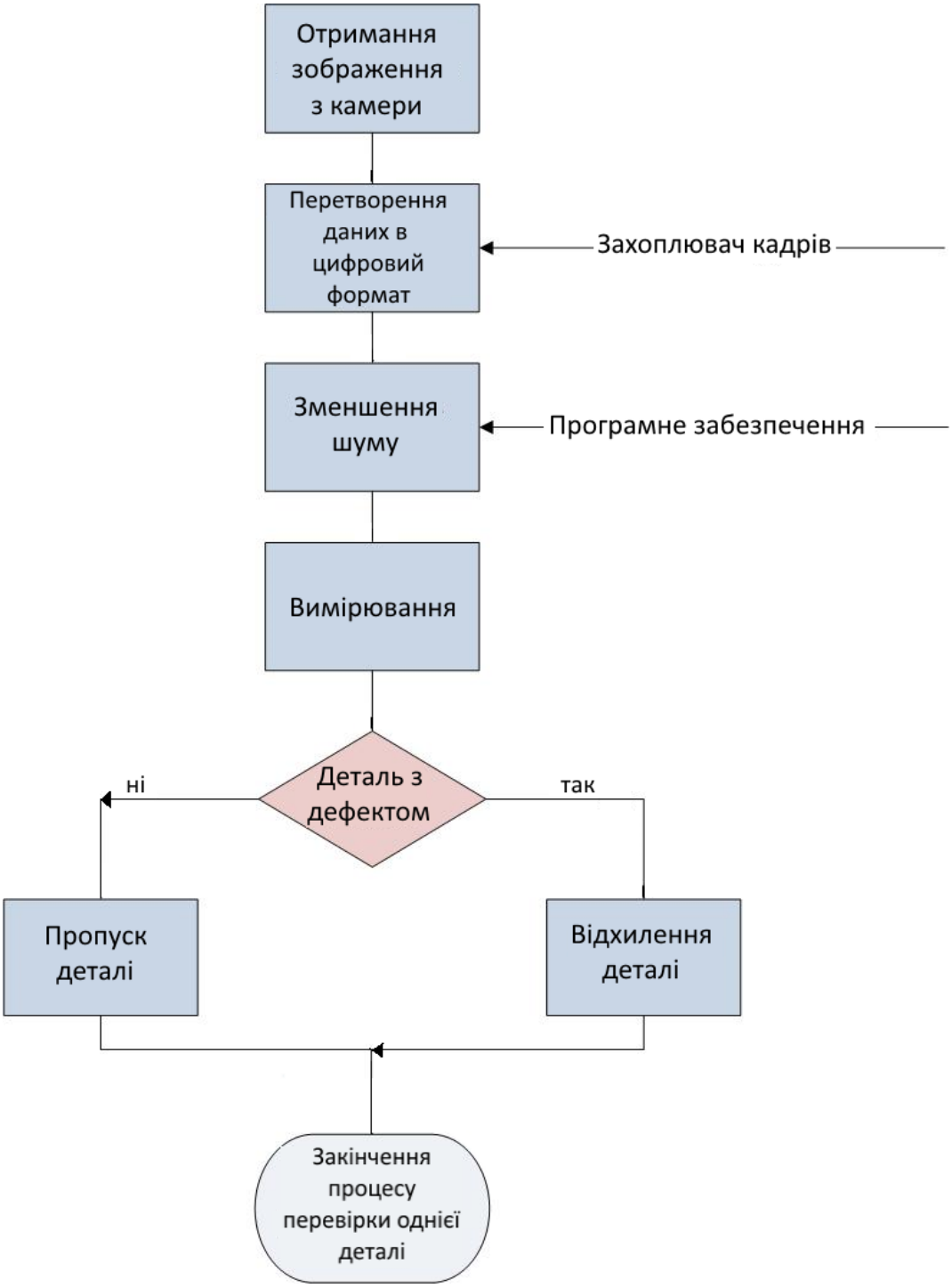


Рисунок 1.4 - Етапи роботи системи машинного зору

Програмне забезпечення машинного зору зазвичай проходить через декілька етапів обробки зображення. Зазвичай, зображення спочатку піддається фільтрації для зниження шуму або перетворюється з різних відтінків сірого у просте чорно-біле зображення (процес бінаризації). На наступному етапі, програма аналізує зображення, виконуючи вимірювання, розпізнавання об'єктів та визначення їх розмірів, дефектів, та інших характеристик. В останньому кроці, вона приймає рішення про прийняття або відхилення деталі відповідно до встановлених критеріїв. У разі виявлення браку, програма може активувати механічний пристрій для відхилення непридатної деталі або зупинити виробничу лінію та сигналізувати оператору про проблему та її причини. Хоча раніше машинний зір частіше використовував монохромні камери, зараз відбувається зростання популярності кольорових камер. Також все більше систем переходять на використання цифрових камер із прямим підключенням, відмовляючись від окремих захоплювачів кадрів, що дозволяє знизити витрати та спростити конструкцію системи.

Машинний зір є частиною інженерії автоматизованих візуалізаційних систем у промисловості і виробництві, інтегруючи різні аспекти комп'ютерних наук, такі як комп'ютерний зір, керувальне устаткування, бази даних, мережеві системи та машинне навчання. Варто зазначити, що комп'ютерний зір є ширшою науковою областю, тоді як машинний зір зосереджений на інженерних рішеннях для виробничих завдань.

Давайте детальніше розглянемо суміжні галузі науки та їх взаємодію з машинним зором. Комп'ютерний зір - це наукова дисципліна, яка фокусується на теоріях та основних алгоритмах для аналізу зображень і сцен. Машинний зір, з іншого боку, є більш всебічною та технічною сферою, що включає повний спектр проблем, пов'язаних із розробкою практичних систем. Це охоплює вибір схем освітлення для аналізованих сцен, характеристики і кількість датчиків, їхнє розміщення, питання калібрування та орієнтування, розробку обладнання для зчитування зображень і обробки даних, а також створення спеціалізованих алгоритмів та їх реалізація.

Існує також область, звана зором роботів. Це специфічна частина технологій машинного зору, що зосереджена на забезпеченні функціонування систем в умовах обмеженого часу. Прикладом є роботи з мобільними камерами та алгоритмами стереозору, що використовуються для створення розумних роботів, здатних самостійно орієнтуватися у просторі, розпізнавати людей та виконувати завдання за допомогою дистанційно поданих команд.

Обробка зображень охоплює будь-який вид обробки даних, де вхідною інформацією є зображення, такі як фотографії чи відеокадри. Цей процес може бути спрямований як на отримання нового зображення, так і на вилучення іншої інформації, наприклад, для розпізнавання тексту або аналізу об'єктів на мікроскопічних знімках. Особливу увагу слід приділити також обробці динамічних зображень, як-от відео. В останні часи термін "обробка зображень" все частіше використовується не як назва наукової дисципліни, а як вказівка на конкретну область застосування. Існує тенденція використання цього терміна для позначення процесів нижчого рівня, де результатом є знову зображення, в той час як "розуміння зображень" стосується процесів вищого рівня.

Цифрова фотограмметрія, що активно розвивається в останні роки, замінила аналітичну фотограмметрію. Відмінно від традиційної аналітичної фотограмметрії, яка фокусувалася переважно на вивченні метричних відносин між елементами знімків та реальних сцен, сучасна цифрова фотограмметрія зосереджується на складніших завданнях, таких як аналіз і 3D-моделювання сцен на основі відеоданих від оптичних сенсорів. Цей напрямок концентрується на точних вимірюваннях елементів сцени та реконструкції форм тривимірних поверхонь за допомогою стереозйомки, мультікамерної зйомки, а також застосування спеціалізованого структурованого освітлення.

Машинне навчання, ключовий сегмент штучного інтелекту, досліджує методики розробки алгоритмів, які здатні самонавчанню. Ця галузь прагне до часткової чи повної автоматизації складних професійних задач у різних сферах діяльності людини. Машинне навчання має широке застосування, включаючи область розпізнавання зображень, що є важливою частиною машинного зору.

1.3 Існуючі алгоритми аналізу шрифтів

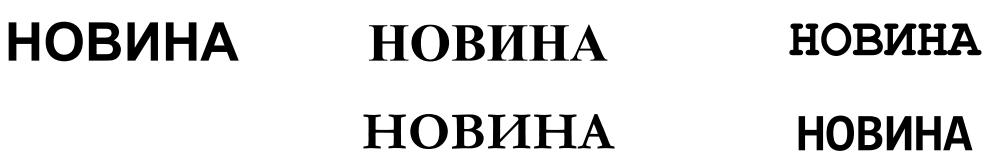
Шрифти, які використовуються в сучасних та старих виданнях, значно відрізняються через еволюцію дизайну шрифтів та технологій друку. Ось детальний огляд типів шрифтів в різних епохах та проблем, які вони можуть викликати при OCR. Сучасні книги та журнали використовують шрифти:

1. Беззасічкові шрифти (англ. Sans-Serif). Часто використовуються для заголовків та тексту в журналах та книгах, що вимагають чистоти та сучасного вигляду. Приклади включають Arial, Helvetica, та Futura.

2. Засічкові шрифти **(**англ. Serif**).** Популярні для основного тексту в книгах, оскільки засічки на кінцях букв допомагають очам легше слідувати за рядками. Приклади включають Times New Roman, Garamond, та Baskerville.

3. Моноширинні Шрифти: Часто використовуються в технічних книгах або документації, де важливо, щоб кожен символ займав однакову ширину. Courier – популярний приклад.

4. Скриптові та декоративні шрифти: Час від часу зустрічаються в заголовках або спеціальних розділах, але рідко використовуються для основного тексту.



Беззасічкові шрифти | Засічкові шрифти | Моноширинні шрифти

Рисунок 1.5 – Типи шрифтів

Старі книги віком більше 100 років використовують:

- Готичні та старослов'янські шрифти: Часто використовувались в Європі до кінця середньовіччя. Характеризуються складними, декоративними формами.

- Гуманістичні та ренесансні засічкові шрифти: Виникли в епоху Відродження, вони більш читабельні та елегантні. Приклади включають шрифти, схожі на Garamond та Caslon.

- Транзиційні та неокласичні шрифти: Розвинулись у 18-19 століттях, ці шрифти мають більш регулярні пропорції та чіткіші засічки.

Проблеми різних типів шрифтів в OCR:

- Складність декоративних шрифтів. Декоративні та готичні шрифти можуть викликати проблеми через їхню складність та незвичність форм букв.

- Нерівномірне чорнило в стародруках**.** Нерівномірний розподіл чорнила та зношення сторінок у старих книгах можуть ускладнювати розпізнавання тексту.

- Нестандартні форми букв. Шрифти старих книг часто мають нестандартні форми букв, що можуть бути невідомі сучасним системам OCR.

- Розмитість та знос сторінок. Фізичний стан старих книг може включати знос, плями, розмитість тексту, що викликає додаткові труднощі при OCR.

- Відсутність контрасту. В старих книгах іноді бракує чіткого контрасту між текстом та фоном, що ускладнює розпізнавання.

У цілому, використання різних типів шрифтів і стан старих книг створюють унікальні виклики для OCR, що вимагає використання спеціалізованих технік та підходів для ефективного розпізнавання тексту.

Детекція назв і типів шрифтів у задачі оптичного розпізнавання символів (OPC) є складним завданням, що включає в себе розпізнавання тексту та аналіз візуальних характеристик шрифту. Ось кілька підходів та відомих наукових джерел у цій області:

1. Машинне навчання та нейронні мережі**.** Сучасні техніки, особливо глибоке навчання, є ефективними у розпізнаванні шрифтів. Конволюційні нейронні мережі (CNN) часто використовуються для аналізу зображень тексту, а класифікація шрифтів може бути реалізована за допомогою багатокласової класифікації.

2. Функціональні характеристики шрифту. Деякі дослідження фокусуються на виявленні унікальних характеристик шрифту, таких як товщина ліній, кут нахилу, відстань між символами тощо, що допомагає у визначенні типу шрифту.

3. Пошук шаблонів та відповідність (matching). Традиційні методи можуть включати в себе використання шаблонів для порівняння та визначення шрифтів. Цей підхід може бути обмеженим через велику різноманітність шрифтів.

4. Трансферне навчання**:** Використання попередньо навчених моделей, які налаштовуються на конкретні завдання детекції шрифтів, також може бути ефективним підходом.

5. Комбінація OCR та аналізу шрифтів**.** Інтеграція систем OCR з алгоритмами аналізу шрифтів може підвищити точність та ефективність розпізнавання тексту та шрифтів.

Питанням розпізнавання документів присвячені спеціалізовані конференції: International Conference on Document Analysis and Recognition (ICDAR). Розглянемо наукові джерела по темі.

Автори [29] описують дослідження у сфері мовного моделювання з використанням нейронних мереж, зокрема, архітектури Long Short-Term Memory (LSTM). Автори починають з констатації, що нейронні мережі стають все більш популярними для завдань мовного моделювання. Мовне моделювання - це процес створення моделей, які можуть передбачати наступне слово у послідовності тексту на основі попередніх слів.

1. Обмеження зворотних нейронних мереж (RNN)**.** Автори зазначають, що традиційні зворотні нейронні мережі (RNN) концептуально можуть враховувати всі попередні слова у послідовності, на відміну від прямих нейронних мереж, які обмежуються фіксованою кількістю попередніх слів. Однак, вони також зазначають, що RNN складно тренувати, що обмежує їхній потенціал.

2. Роль архітектури LSTM. Автори вказують, що проблеми, пов'язані з тренуванням RNN, вирішуються за допомогою архітектури LSTM. LSTM була розроблена для вирішення проблеми "зникнення градієнтів" в RNN, що робить її ефективнішою для довготривалого зберігання інформації.

3. Автори описують застосування LSTM до завдань мовного моделювання для англійської та французької мов. Це вказує на те, що вони використовували LSTM для створення моделей, які можуть передбачати наступні слова в тексті цих мов.

4. Автори зазначають, що експерименти показали поліпшення відносно традиційних RNN на приблизно 8% за мірою "perplexity" (заплутаність), яка є стандартною метрикою в оцінці мовних моделей. Крім того, вони відзначають значне покращення у показнику помилок розпізнавання мови (WER) при використанні на вершині системи розпізнавання мовлення останнього покоління.

Приклади зображень з [29] для навчання аналізатора стильових ознак на рисунку 1.6.

Рисунок 1.6 - Приклади зображень для навчання аналізатора стильових ознак

В результаті досліджень автори виділили які типи шрифтів найчастіше зустрічаються на різних типах об’єктів на реальних сценах.

Таблиця 1.1 – Комбінація написів на об’єктах та шрифтів

Об’єкт	Стиль шрифту
Книга	Serif (з засічками)
Автобус	Sans-serif конденсований
Дошка	Sans-serif із вузьким кернінгом
Мобільний телефон	Рукописний
Дорожній знак	Sans-serif (Без засічок)
Телевізор	Sans-serif (Без засічок)
Монітор комп'ютера	Sans-serif (Без засічок)

При аналізі друкованих книжок автори [29] виявили наступне:

- слова, пов'язані з релігією, як правило, пишуться шрифтом із засічками.
- слова, пов'язані з IT, як правило, пишуться без засічок.
- слова, пов'язані з аристократією, як правило, пишуться із засічками.
- хоча слова, пов'язані з пресою, охоплюють широкий спектр шрифтів, вони відносно часто пишуться із засічками.
- слова, що використовуються в гостросюжетних романах, такі як "вбити", "самогубство", "вбивство", пишуться без засічок, ущільнено.

- слова, пов'язані з війною, часто пишуться без засічок, так само як і в гостросюжетних історіях.
- слова, пов'язані з хімією та медициною, як правило, пишуться шрифтами із засічками та без них.

Метод авторів [29] полягає в тому що при розпізнаванні тексту вони визначають класи зображень і об’єктів на поверхні яких є текст. Із таких міток чи категорій створюють новий простір ознак який потім кластеризують.

Автори [28] описують дослідження, спрямоване на аналіз взаємозв'язку між стилями шрифтів та контекстуальними факторами, які можуть впливати на вибір шрифту. Автори вказують на існування численних стилів шрифтів у світі, які створюють різні враження та мають різну читабельність. Основна мета дослідження полягає у вивченні взаємодії між стилями шрифтів та різними контекстуальними факторами. Це робиться шляхом аналізу великих наборів даних. Для аналізу автори використовують приблизно 800,000 слів із набору даних Open Images. Вони досліджують зв'язок між стилем шрифту та об'єктами, що його оточують (наприклад, слово "автобус" та шрифт, який використовується для його написання). Також використовується набір даних обкладинок книг для аналізу зв'язку між стилями шрифтів та жанрами книг. Додатково розглядається семантичне значення слова як один з факторів, що впливають на вибір шрифту. Для кількісного аналізу використовуються власна модель вилучення ознак стилів шрифтів та word2vec, метод для аналізу семантичних відносин між словами. В результаті аналізу автори знаходять кілька прикладів, коли певні стилі шрифтів використовуються для конкретних контекстуальних факторів.

1.3 Засоби розпізнавання тексту

Сучасні бібліотеки оптичного розпізнавання символів (OCR), що можуть бути інтегровані з Python, включають Tesseract, Pytesseract, EasyOCR, OCRopus, та інші. Ці бібліотеки використовуються для перетворення зображень тексту

(наприклад, сканованих документів або фотографій) у машиночитабельний текст. Давайте розглянемо докладніше кожну з них.

Tesseract OCR Tesseract - це відкритий проект OCR, спочатку розроблений HP, а потім Google. Це одна з найбільш точних безкоштовних бібліотек OCR і підтримує більше 100 мов, включаючи кирилицю. Починаючи з версії 4.0, використовує архітектуру LSTM (Long Short-Term Memory), яка є однією з форм глибокого навчання для вдосконалення точності розпізнавання тексту. LSTM дозволяє Tesseract ефективніше обробляти послідовності даних, такі як текст, оскільки вона краще справляється з контекстною інформацією та залежностями між символами. Переваги:

- висока точність і надійність.
- підтримка великої кількості мов.
- гнучкість: можливість тренування для специфічних шрифтів або мов.
- активна спільнота і постійні оновлення.

Недоліки:

- Потребує налаштування для максимальної точності.
- Може бути складним у використанні для новачків.

Pytesseract - це обгортка Python для Tesseract OCR, що дозволяє легко інтегрувати його функціонал в Python-проекти. Переваги**:**

- Легке інтегрування з Python.
- Всі переваги Tesseract OCR (висока точність, підтримка багатьох мов).
- Простота використання.

Недоліки:

- Залежить від встановлення Tesseract OCR на машині користувача.
- Точність залежить від якості вхідного зображення.

EasyOCR - це відносно нова бібліотека, яка базується на глибокому навчанні і підтримує понад 60 мов, включаючи кирилицю. Переваги:

- Легке використання.
- Підтримка багатьох мов.
- Добра точність розпізнавання.

Недоліки:

- Може мати проблеми з розпізнаванням складних макетів.

- Вимагає значних обчислювальних ресурсів.

OCRopus - це OCR-система з відкритим кодом, яка також використовує методи глибокого навчання. Переваги:

- Висока точність для чітких зображень.

- Підтримка кирилиці.

Недоліки:

- Складність у встановленні та налаштуванні.

- Менш популярна, порівняно з Tesseract.

Таблиця 1.2 - Порівняльна таблиця засобів розпізнавання

Характеристика	Tesseract OCR	Pytesseract	EasyOCR	OCRopus
Точність	Висока	Висока	Добра	Висока
Легкість використання	Потребує налаштувань	Легке	Дуже легке	Складне
Підтримка мов	>100 мов	Залежить від Tesseract	>60 мов	Так, включаючи кирилицю
Методи	LSTM	LSTM	Глибоке навчання	Глибоке навчання
Ресурсоємність	Середня	Середня	Висока	Середня
Спільнота	Активне	Менше ніж Tesseract	Розвивається	Менше ніж Tesseract

Вибір бібліотеки OCR залежить від конкретних потреб проекту, якості вхідних даних та необхідної точності розпізнавання. Для більшості випадків Tesseract або Pytesseract є достатньо потужними і гнучкими, тоді як EasyOCR може бути кращим варіантом для проектів, які потребують швидкого розгортання без складних налаштувань.

1.4 Аналіз завдання магістерської роботи та постановка задач дослідження

Систематизуємо різні види спотворень символів у стрічках тексту які виникають під час оптичного розпізнавання символів. Існують різні способи класифікації спотворень, залежно від джерела (сканування, друк, рукопис), типу (геометричні, фотографічні, текстові) або впливу на текст (розмиття, злипання, відсутність елементів).

Спотворення спричинені скануванням або фотографуванням:

- Розмиття. Текст може виглядати розмитим через низьку якість сканування або рух камери.

- Спотворення перспективи. Текст може бути спотворений, якщо документ був сфотографований під кутом.

- Варіації освітлення. Нерівномірне освітлення може спричинити відтінки або відблиски на тексті.

Спотворення спричинені якістю документу:

- Зношеність або пошкодження**.** Старі або пошкоджені документи можуть мати нечіткі або відсутні частини тексту.

- Чорнильні розтікання. Розтікання чорнила може спричинити злипання символів.

Спотворення специфічні для тексту:

- Злипання символів: Символи можуть дотикатися один до одного, ускладнюючи їх розпізнавання.

- Перекриття символів: Символи можуть перекривати один одного, особливо в рукописних текстах.

- Різноманітність шрифтів**:** Різні шрифти та розміри можуть впливати на точність OCR.

Методи подолання спотворень. Методи попередньої обробки:

- Бінаризація: Перетворення зображення в чорно-біле для підвищення контрасту.

- Видалення шуму: Використання фільтрів для видалення непотрібних елементів.

- Корекція перспективи і вирівнювання: Виправлення спотворень, викликаних кутом зйомки.

- Покращення контрасту і освітлення досягнення чіткості.

Таблиця 1.3 - Види спотворень тексту та методи їх виправлення

Тип спотворення	Опис	Методи подолання
Спричинені скануванням або фотографуванням		
Розмиття	Нечіткість тексту через низьку якість сканування або рух камери.	Бінаризація, видалення шуму, покращення контрасту.
Спотворення перспективи	Текст спотворений через кут зйомки.	Корекція перспективи і вирівнювання.
Варіації освітлення	Нерівномірне освітлення створює відтінки або відблиски.	Покращення контрасту і освітлення.
Спричинені якістю документу		
Зношеність або пошкодження	Пошкоджені частини тексту на старих документах.	Видалення шуму, післяобробка з використанням мовних моделей.
Чорнильні розтікання	Злипання символів через розтікання чорнила.	Розбиття злиплих символів, конволюційні нейронні мережі.
Специфічні для тексту		
Злипання символів	Символи дотикаються один до одного.	Розбиття злиплих символів, конволюційні нейронні мережі.
Перекриття символів	Символи перекривають один одного.	Розбиття злиплих символів, глибоке навчання.
Різноманітність шрифтів	Різні шрифти та розміри утруднюють розпізнавання.	Конволюційні нейронні мережі, рекурентні нейронні мережі.

Методи сегментації тексту:

- Розбиття тексту на рядки і слова: Використання алгоритмів для ідентифікації окремих рядків і слов.

- Розбиття злиплих символів: Застосування спеціалізованих алгоритмів для виявлення і розділення злиплих символів.

Застосування глибокого навчання:

- Конволюційні нейронні мережі (CNN): Для виявлення і класифікації символів в умовах різних спотворень.

- Рекурентні нейронні мережі (RNN): Особливо ефективні для розпізнавання послідовностей символів у тексті.

Методи післяобробки:

- Використання словників і мовних моделей: Для виправлення помилок та відновлення відсутніх частин тексту.

- Оцінка ймовірності**:** Визначення найімовірніших символів або слів на основі контексту.

Математичний апарат деяких методів комп'ютерного зору та штучного інтелекту, які використовуються для аналізу та корекції різних видів спотворень, зокрема у контексті OCR.

Метод Оцу використовується для бінаризації зображення. Він автоматично вибирає порогове значення, щоб мінімізувати ваговану варіативність всередині класів (чорних та білих пікселів). Математично, він максимізує вираз:

$$\sigma_B^2(t) = \omega_0(t)\omega_1(t)[\mu_0(t) - \mu_1(t)]^2$$

де σ^2_B - це міжкласова варіативність,

ω_i - ймовірності класів,

μ_i - середні значення класів.

Конволюційні нейронні мережі (CNN) використовують фільтри для видобування особливостей із зображення. Конволюція, основна операція в CNN, математично визначається як:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau)d\tau$$

де f і g - функції, що представляють вхідне зображення та ядро фільтра.

Рекурентні нейронні мережі (RNN) ефективні для роботи з послідовностями даних. Вони використовують зворотні зв'язки для збереження інформації про попередній стан. Комірка RNN описується формулою:

$$h_t = \sigma(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$

де h_t - прихований стан,

x_t - вхідний вектор,

W - вагові матриці,

b - вектор зміщення.

Гаусівське розмиття використовує ядро, елементи якого є взаємно корелюючими за Гаусівською функцією:

$$G(x,y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

де G - Гаусівське ядро,

σ - стандартне відхилення (контролює ступінь розмиття).

Оператор Собеля використовується для виявлення країв у зображенні. Він обчислює градієнт яскравості зображення за допомогою фільтрації:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A$$
$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * A$$

де A - вхідне зображення,

G_x і G_y - горизонтальні та вертикальні компоненти градієнта.

Морфологічні операції. Дилатація та ерозія - основні морфологічні операції, які використовуються для зміни форми об'єктів на зображенні. Вони визначаються як:

Дилатація: $(A \oplus B) = \{z|(\hat{B})_z \cap A \neq \emptyset\}$

Ерозія: $(A \ominus B) = \{z|B_z \subseteq A\}$

де A - вхідне зображення, B - структурний елемент.

Узагальнені етапи алгоритму.

1. Виявлення та вибір літер шрифту:

- Розробка або інтеграція OCR-модуля для виявлення тексту на зображенні.

- Розробка методу для ідентифікації та відокремлення літер певного шрифту з тексту на зображенні.

2. Векторизація літер:

- Розробка або використання існуючого інструменту для перетворення растрових зображень літер у векторний формат.

- Забезпечення точності векторизації для збереження ключових особливостей шрифту.

3. Створення TTF шрифту:

- Розробка або інтеграція інструменту для генерації TTF файлу шрифту на основі векторизованих літер.

- Врахування стандартів та вимог до TTF шрифтів для забезпечення сумісності.

4. Генерація тексту та виправлення помилок:

- Інтеграція створеного шрифту в OCR-систему для генерації тексту.

- Розробка алгоритму для виправлення помилок, виявлених на етапі виявлення літер.

Технічні Вимоги до ПЗ:

- Визначення використовуваного програмного забезпечення, мов програмування Python, бібліотек та фреймворків.

- Забезпечення високої точності у виявленні літер та векторизації.

- Оптимізація алгоритму для швидкої та ефективної обробки.

Тестування та валідація:

- розробка тестових сценаріїв для перевірки всіх компонентів алгоритму.

- проведення випробувань для визначення точності, швидкості обробки та надійності алгоритму.

- валідація роботи алгоритму на різних типах зображень та шрифтів.

1.5 Висновки до розділу

Систематизовано види спотворень символів у стрічках тексту. Вони класифікуються, залежно від джерела (сканування, друк, рукопис), типу (геометричні, фотографічні, текстові) або впливу на текст (розмиття, злипання, відсутність елементів). Сформовано технічне завдання на розроблення програмного забезпечення.

2 РОЗРОБКА АЛГОРИТМІВ РЕКОНСТРУКЦІЇ ШРИФТІВ

2.1 Алгоритми реконструкції на основі контурів

У процесі розпізнавання тексту, який зображено на ілюстрації 1.2, використовуються різноманітні процедури та техніки, включаючи:

- підготовча обробку;
- сегментація символів;
- процес розпізнавання.

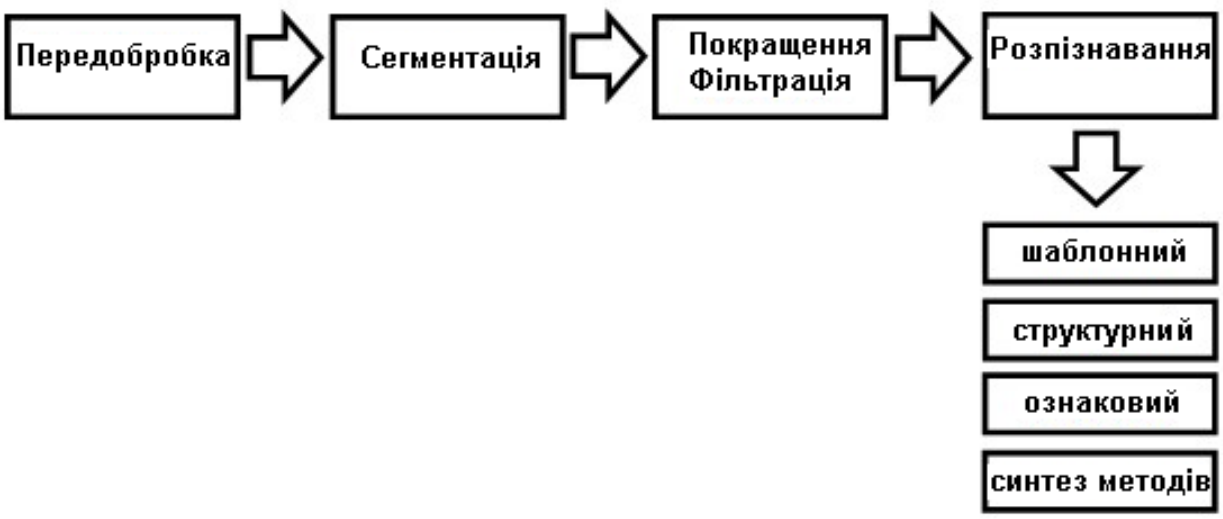


Рисунок 2.1 - Основні етапи та методи розпізнавання тексту

Початковий етап, підготовча обробка, є стандартною практикою в текстовому аналізі. Цей етап включає застосування технік усереднення, вирівнювання гістограм, а також застосування різних фільтрів для мінімізації перешкод і зниження впливу зовнішніх шумів.

Сегментація передбачає розділення зображення на індивідуальні символи. Останнім кроком є власне розпізнавання, де використовуються зображення, очищені від шуму та сегментовані.

Існують три основні підходи до розпізнавання символів: шаблонний, структурний та ознаковий.

Шаблонні методи полягають у перетворенні зображення символу в растровий формат та порівнянні його із шаблонами у базі, вибираючи той, що має найменше розходжень. Вони ефективні проти дефектів зображення і швидкі в обробці, але обмежені відомими шрифтами. Шаблонні методи схильні до помилок, якщо розпізнаваний шрифт навіть незначно відрізняється від зразка в базі, навіть при високій якості зображень.

В структурному підході до розпізнавання символів, об'єкт репрезентується у вигляді графа, де його елементи виступають як вершини, а їх просторові зв'язки - як ребра. Цей метод, як правило, використовує векторні зображення і фокусується на лінійних складових символу. Наприклад, буква "р" аналізується як комбінація вертикального сегмента і дуги. Однак, ці методи чутливі до пошкоджень зображення, які можуть порушити структурні елементи, і векторизація може вносити додаткові дефекти. На відміну від шаблонних і ознакових методів, для структурних методів часто бракує ефективних процедур автоматичного навчання, тому структурні описи зазвичай створюються вручну.

У ознакових методах зображення символів представляються у n-мірному просторі характеристик. Для цього вибираються певні ознаки, значення яких обчислюються при аналізі вхідного зображення. Вектор ознак порівнюється з еталонними векторами, і зображення класифікується за найбільш схожим.

Існують також методи, що поєднують різні підходи, і ці методи часто виявляються ефективними на практиці. Один з таких методів - розпізнавання скелетних зображень. Цей процес починається зі скелетизації символу, де зображення зведене до свого скелету за допомогою різних методів. Один з таких методів - метод Щепіна, який полягає в ідентифікації вихідних точок на контурах зображення, аналізі сусідніх точок та видаленні тих, які не є кінцевими і не порушують зв'язність. Цей процес повторюється до тих пір, поки не залишаться тільки скелетні точки (див. рисунок 2.2).

Процес скелетизації за допомогою використання спеціалізованих шаблонів (паттернів) включає видалення надлишкових пікселів із зображення, використовуючи шаблони, де «X» позначає пікселі будь-якого кольору (див. рисунок 2.3). Цей метод полягає у видаленні центрального чорного пікселя в

будь-якій зоні, яка відповідає заданому шаблону. Процедура повторюється багаторазово на всьому зображенні до тих пір, поки не залишаться лише ті пікселі, які не відповідають критеріям видалення. Цей підхід дозволяє точно і ефективно відтворити скелетне зображення, забезпечуючи видалення лише непотрібних елементів.

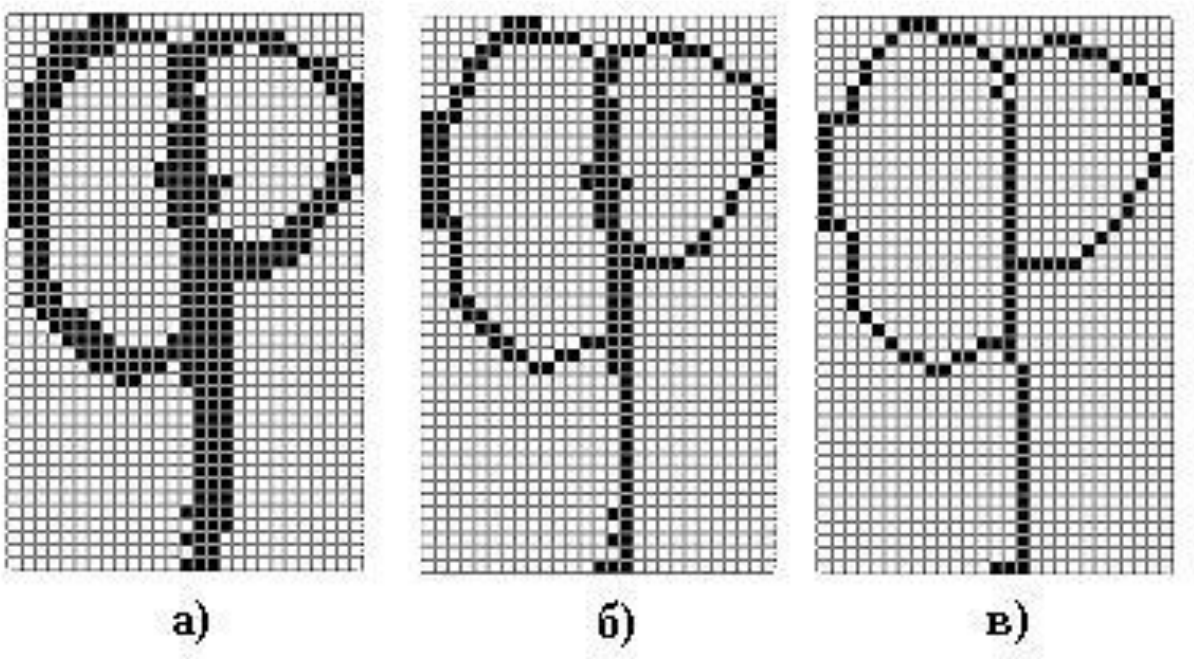


Рисунок 2.2 − Процес скелетизації літери «Ф», що включає в себе як зовнішній, так і два внутрішніх контури.
а) Початкове зображення; б) Видалення першого рівня контурів; в) Видалення другого рівня контурів.

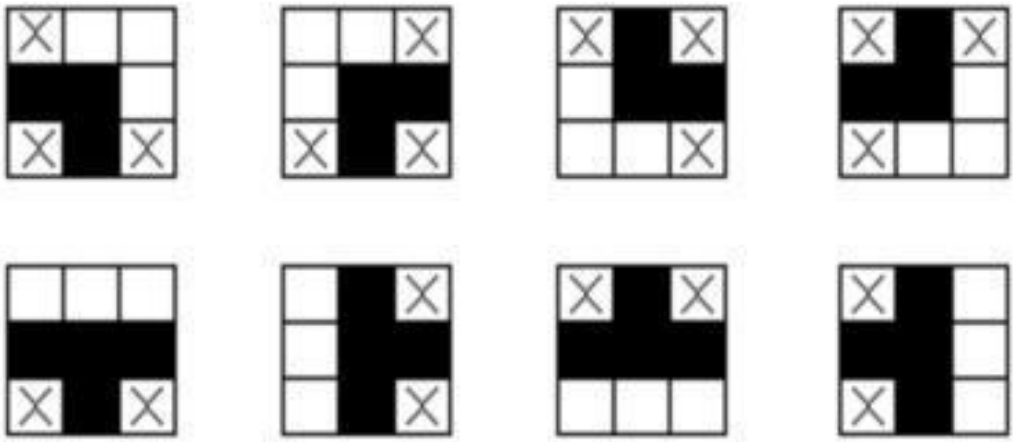


Рисунок 2.3 − Паттерни скелетизації

В процесі підготовки скелетного зображення до етапу розпізнавання, ми виконуємо визначення ключових точок. Це включає в себе точки, де зустрічаються три або чотири ребра скелета, а також кінцеві точки:

1. Створюємо пустий стек для зберігання даних про вихідні та кінцеві точки ребер, а також точки розгалужень скелета.

2. Додаємо у стек випадкову точку зі скелета.

3. Продовжуємо обробку, доки стек не стане порожнім, виконуючи кроки з 4 по 7.

4. Витягуємо точку із стека.

5. Слідуємо вздовж ребер від вибраної точки до наступної точки розгалуження скелета або до кінцевої точки.

6. У випадку досягнення кінцевої точки або раніше позначеного ребра, записуємо шлях у масив.

7. При досягненні точки розгалуження скелета (точка з'єднання ребер) фіксуємо ребра у масиві і додаємо точку розгалуження у стек.

8. Повторюємо від кроку 3.

У фінальному описі скелета застосовується груба попередня обробка, яка включає усунення коротких відрізків і злиття близько розташованих точок розгалуження

Метод розповсюдження хвилі включає аналіз траєкторії розповсюдження сферичної хвилі через зображення (див. рисунок 2.4). В основі цього методу лежить оцінка зміни центру мас пікселів, які формують нову фронт хвилі, відносно їх попереднього розташування. Процедура складається з таких кроків:

- Створення скелета зображення за допомогою розповсюдження сферичної хвилі;

- Оптимізація сформованого скелета.

Процес відстеження ліній у зображенні здійснюється шляхом спостереження за переміщенням центру мас відрізка, що утворений крайніми точками фронту хвилі (див. рисунок 2.5, а). Після відстеження, лінії можуть бути піддані додатковому згладжуванню.

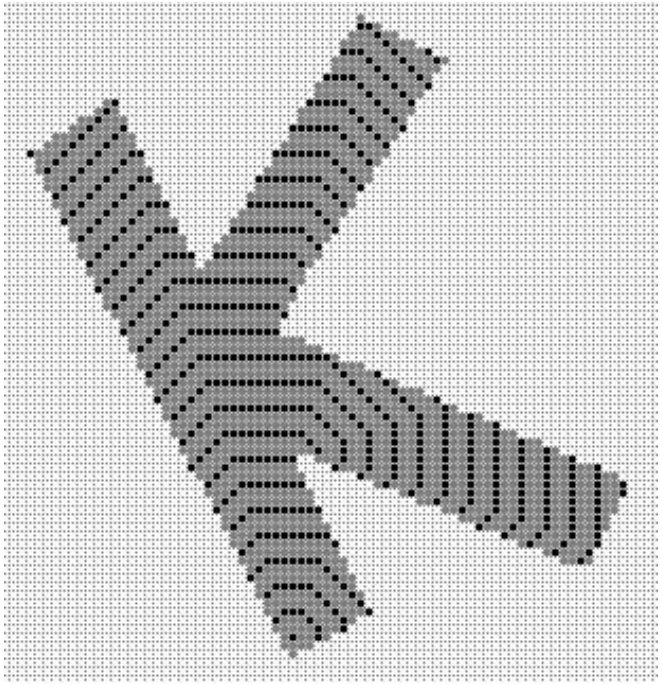


Рисунок 2.4 - Поширення сферичної хвилі через зображення.

Процес виявлення збільшення «ширини» хвилі та її розподілу на менші сегменти допомагає визначити місце, де потенційно можуть з'єднатися два відрізки. Оцінка зміни «ширини» хвилі відбувається через порівняння ширини поточної генерації хвилі з її середнім значенням за N попередніх генерацій (де N визначається заздалегідь). У результаті, отримуємо дві точки (A і B), що формують трасований відрізок. Далі, коли хвиля розділяється на дві половини, виокремлюються додаткові пари точок (C і D) та (E і F). Точка, де з'єднуються відрізки, розташована всередині шестикутника ABCDEF і спочатку визначається як центр мас цього багатокутника (див. рисунок 2.5, б). Процедура корекції та оптимізації скелета зображення виконується в останньому етапі.

Сформований скелет зображення часто виявляється недосконалим, особливо через обмеження, що накладаються растровим форматом зображення. Такі обмеження ведуть до більшого спотворення, особливо при малих розмірах зображення.

Для мінімізації впливу цих спотворень на кінцевий скелет потрібно провести його оптимізацію. В процесі оптимізації виявляється, що один відрізок може бути представлений як ланцюг ребер. Це можна виправити, аналізуючи послідовність ребер та оцінюючи, наскільки отримана лінія наближена до

прямої. Якщо відхилення виявляється припустимим, ланцюг ребер замінюється одним відрізком.

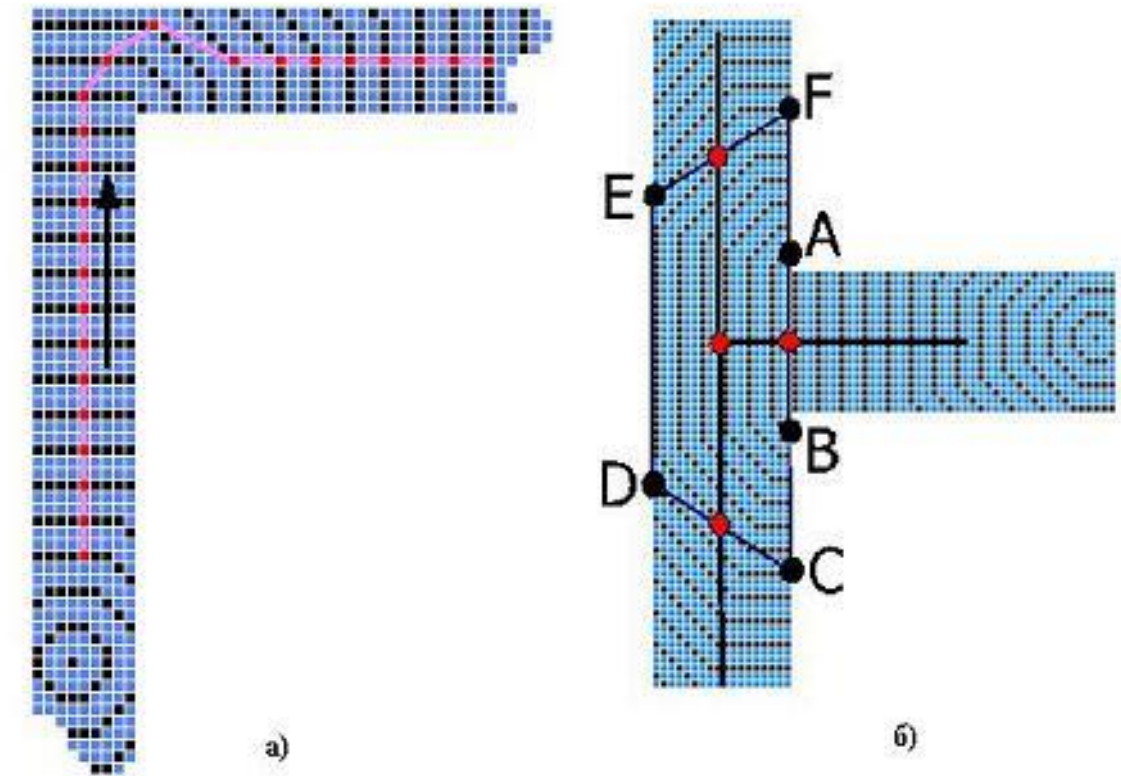


Рисунок 2.5 а) процес відстеження ліній у зображенні,
б) локалізацію точок, де з'єднуються різні сегменти.

Оптимізація скелета також включає аналіз околиць визначених точок з'єднання відрізків, де спостерігається розділення хвилі на дві частини.

Часто знаходжувані викривлення (див. рисунок 2.6, а) коригуються шляхом аналізу сегментів, пов'язаних з визначеною точкою (А) (наприклад, AB_1, B_1C_1, AB_2, B_2C_2, AB_3, B_3C_3).

Цей аналіз включає пошук таких пар сегментів C_xB_x та C_yB_y з серії (B_1C_1, B_2C_2, B_3C_3), які утворюють лінії, найбільш наближені до прямої. У результаті, точку (А) необхідно перенести на місце перетину ліній C_xC_y та AC_2. Після цього точки B_1, B_2, B_3 видаляються з графа (див. рисунок 2.6, б), оптимізуючи таким чином структуру скелета зображення.

Інший тип спотворення зображення виникає, коли три лінії зустрічаються в одній точці (показано на рисунку 2.5, в). В такому випадку немає можливості вибрати пару відрізків, які утворюють пряму лінію. Точка (А) має бути

переміщена в центр трикутника, створеного лініями B_1C_1, B_2C_2 і B_3C_3. Після цього точки B_1, B_2, B_3 слід вилучити з графа (див. рисунок 2.6, г).

У контексті алгоритму розпізнавання для кожної ключової точки скелетного представлення обчислюються топологічні характеристики. Основні з них включають:

1. Нормалізовані координати ключової точки (вершини графа).

2. Відносна довжина відрізка до наступної вершини у відсотках від загальної довжини графа.

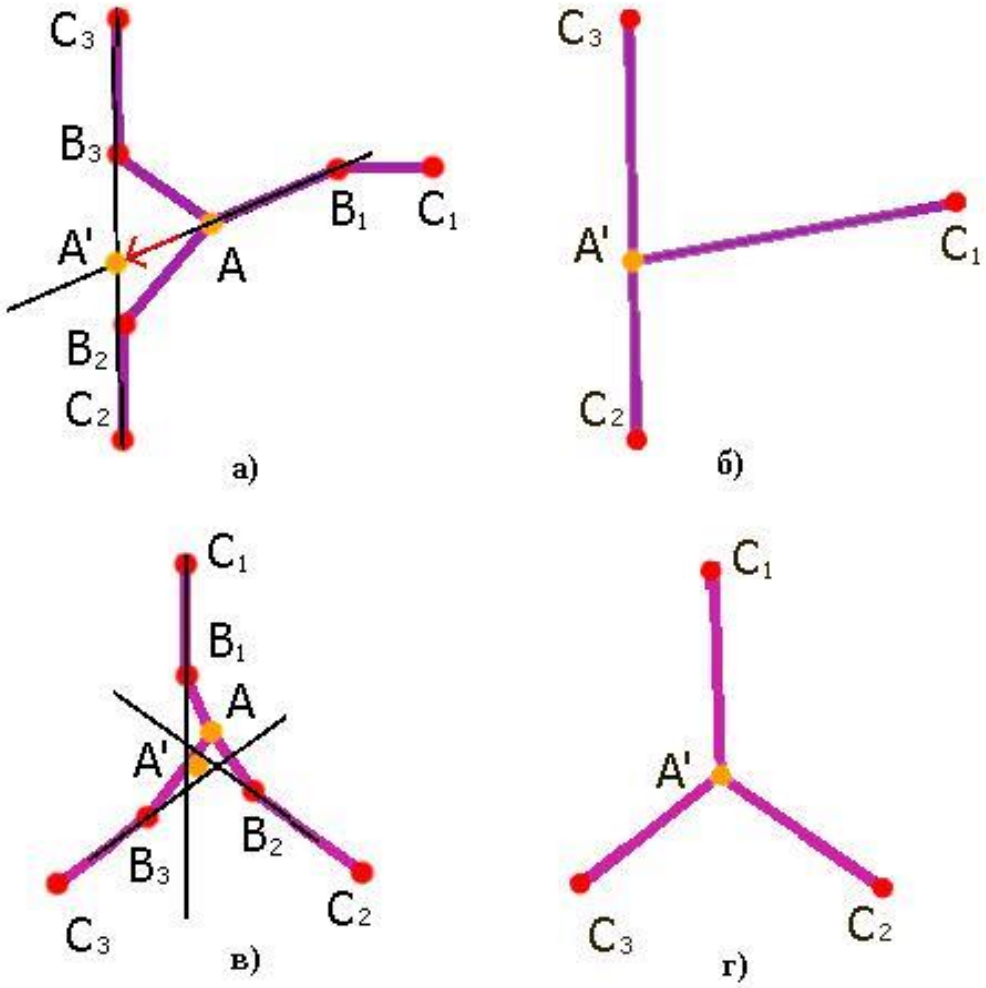


Рисунок 2.6 - Оптимізація точки приєднання відрізків

3. Визначення нормованого напрямку від даної точки до наступної особливої позиції.

4. Визначення нормованого напрямку входу до точки та виходу з неї.

5. Врахування кривизни дуги, а саме - лівої та правої кривизни дуги, що з'єднує ключову точку з наступною вершиною (ліворуч та праворуч). Розрахунок кривизни полягає у визначенні співвідношення максимальної відстані між точками дуги (розташованими відповідно ліворуч та праворуч від прямої) до довжини відрізка, що приєднує ті ж самі вершини, та подальше врахування цього співвідношення.

На зображенні (рисунок 2.7) нам показані концептуальні представлення деяких топологічних властивостей. Граф складається з п'яти особливих точок: a_0, a_1, a_2, a_3, a_4. Під час подорожі по графу, наприклад, від a_0 до a_1, в вершині a1 відображаються наступні характеристики:

вектор R1 - напрямок входу в точку, вектор R2 - напрямок виходу з точки, вектор R3 - глобальний напрямок до наступної особливої точки. Крім цього, двонапрямлений вектор *h* вказує величину "лівого" відхилення дуги (a_1, a_2) від прямої; "праве" відхилення рівне нулю.

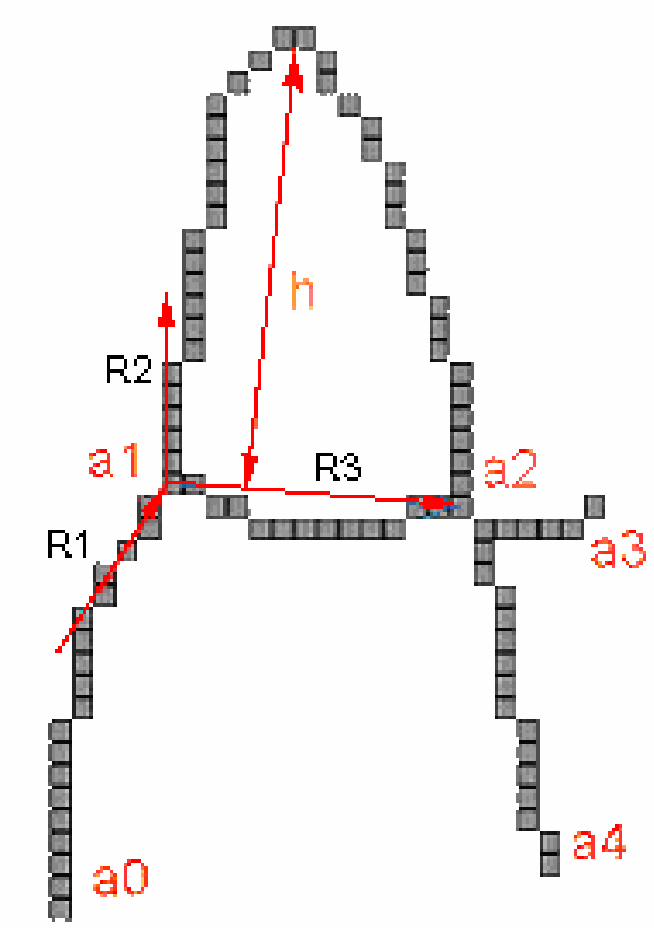


Рисунок 2.7 - Ілюстрація Топологічних Характеристик

Для певних кодів може бути недостатньо особливих точок і, відповідно, топологічних характеристик. Наприклад, для коду, що представляє символ "0", топологічних характеристик зовсім немає через відсутність особливих точок. В таких випадках можуть бути використані та обчислені додаткові характеристики:

1) Розміри та розташування компонентів і отворів.

2) "біла" і "чорна" ширина верхньої половини символу.

3) Модифіковані горизонтальні прямокутні відрізки - прогини.

Обчислення прогинів полягає у вимірюванні відстаней від точок, що належать скелетному представленню, до опуклої оболонки побудованого зображення. Окрім цього, зберігається інформація про положення точок максимального прогину. Для деяких топологічних кодів, кількість характеристик може бути досить значною, що може потребувати великий обсяг навчальних даних для навчання системи. В таких ситуаціях, для розпізнавання може використовуватися лише підмножина з цих характеристик.

Визначення символу відбувається шляхом порівняння його опису з кодами, збереженими в базі даних. Після цього вибирається найближчий топологічний код.

Якщо символ залишається нерозпізнаним після пройденого циклу розпізнавання, застосовуються наступні операції для його поліпшення:

1) З'єднання кінців ліній за напрямками (див. рисунок 2.8); для цього аналізуються напрямки всіх кінцевих дуг скелетону, і якщо напрямки деяких ліній схожі (з урахуванням знаку) і вказують один на одного, то їх можна спробувати з'єднати - можливо, це була єдина лінія, яка розірвалася через недостатню якість сканування або вхідні дефекти в рукописі.

2) З'єднання точок скелета, які знаходяться на мінімальній відстані одна від одної.

3) Видалення найкоротших ліній (дуг) графу; зайві короткі дуги (лінії) часто виникають при написанні рукописному.

Рисунок 2.8 - Покращення Зображення Символу: а) початкове зображення символу
б) символ зі зєднаними лініями.

Після ретельного аналізу різних методів реалізації алгоритму розпізнавання було вирішено застосувати декілька алгоритмів для виділення та розпізнавання символів. Використання шаблонного методу обрано з ряду причин:

- цей метод проявляє стійкість до спотворень даних.
- шаблонний метод володіє високою швидкістю обробки даних.
- існує попередня інформація щодо єдиного можливого шрифту.
- Для збереження еталонів не потрібно великої кількості пам'яті.

2.2 Алгоритм виявлення символів на основі проекції

Алгоритм для проектування стрічки з довільним текстом та створення візуалізації функції проекції.

1. Генерує зображення з довільним текстом заданого типу.
2. Конвертує зображення у бінарний формат.
3. Виконує горизонтальну проекцію.
4. Візуалізує результат проекції.

Імпорт бібліотек.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

Функція створення тексту шрифтом заданого типу. Код дозволяє вибирати різні шрифти та їхні розміри для генерації текстового зображення. Можна змінювати параметри font, font_scale та thickness для досягнення бажаного вигляду тексту.

```
def create_text_image(text, font=cv2.FONT_HERSHEY_SIMPLEX, font_scale=1, thickness=2):
    """ Створює зображення з текстом. """
    # Визначення розміру зображення, необхідного для тексту
    text_size = cv2.getTextSize(text, font, font_scale, thickness)[0]
    text_width, text_height = text_size
    image = np.zeros((text_height + 20, text_width + 20), dtype=np.uint8)

    # Додаємо текст на зображення
    text_org = (10, text_height + 10)
    cv2.putText(image, text, text_org, font, font_scale, (255, 255, 255), thickness, cv2.LINE_AA)

    return image
```

У OpenCV (cv2) доступні такі основні шрифти для використання з функцією cv2.putText:

1. cv2.FONT_HERSHEY_SIMPLEX: звичайний шрифт без засічок.
2. cv2.FONT_HERSHEY_PLAIN: простий шрифт без засічок.
3. cv2.FONT_HERSHEY_DUPLEX: подвійний шрифт без засічок.
4. cv2.FONT_HERSHEY_COMPLEX: складний шрифт без засічок.
5. cv2.FONT_HERSHEY_TRIPLEX: потрійний шрифт без засічок.
6. cv2.FONT_HERSHEY_COMPLEX_SMALL: менший варіант складного шрифту без засічок.
7. cv2.FONT_HERSHEY_SCRIPT_SIMPLEX: шрифт, що імітує рукописний текст.
8. cv2.FONT_HERSHEY_SCRIPT_COMPLEX: більш складний варіант рукописного шрифту.

Щоб використовувати функції OpenCV для роботи зі шрифтами із зовнішніх файлів, потрібно використання додаткових бібліотек freetype.

Функція проекції

```
def horizontal_projection(binary_image):
    """ Проектує бінарне зображення на горизонтальну вісь. """
    return np.sum(binary_image, axis=0)
```

Візуалізація результату

```
def visualize_projection(text_image, projx):
    """ Візуалізує горизонтальну проекцію під зображенням з текстом.
"""
    plt.figure(figsize=(10, 4))

    # Відображення тексту
    plt.subplot(2, 1, 1)
    plt.imshow(text_image, cmap='gray')
    plt.title('Зображення тексту')
    plt.axis('off')
    # Візуалізація проекції
    plt.subplot(2, 1, 2)
    plt.plot(projx)
    plt.title('Горизонтальна проекція')
    plt.xlabel('Позиція')
    plt.ylabel('Сума значень пікселів')
    plt.tight_layout()
    plt.show()
```

Дана функція дозволяє отримати такий результат.

Рисунок 2.9 – Проекція суми пікселів на вісь X

Додамо шум типу «сіль та перець» на зображення .

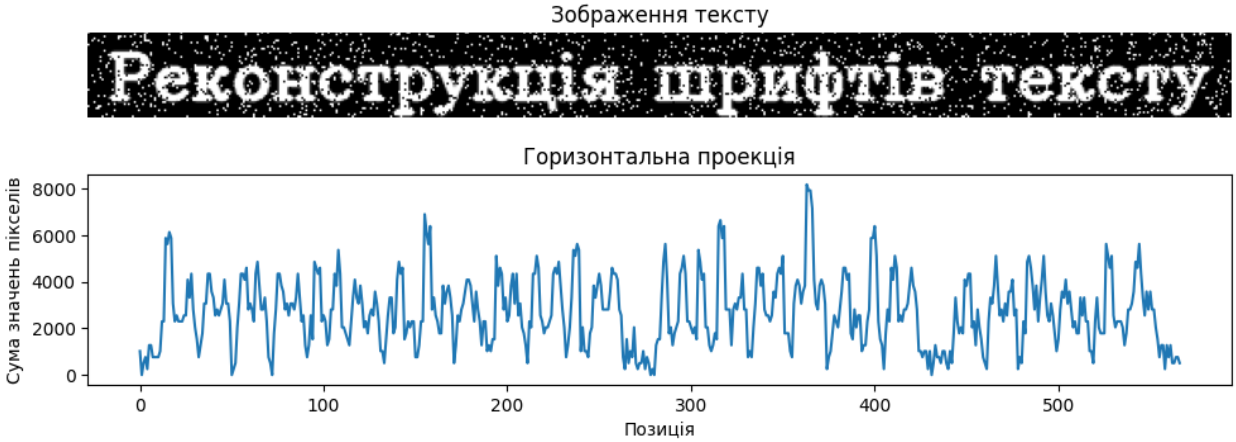


Рисунок 2.10 – Проекція суми пікселів на вісь X для спотвореного зображення

Для визначення позицій пробілів між літерами тексту на основі функції проекції на вісь X, можна використовувати два підходи:

1. Ширина літер. Якщо відома ширина літер, можна шукати пробіли, які відповідають певній мінімальній ширині. Це працює добре для моноширинних шрифтів.

2. Дисперсія. Якщо ширина літер невідома або якщо шрифт не є моноширинним, можна використовувати дисперсію. Метод дисперсії шукає зміну в щільності пікселів в горизонтальній проекції. Пробіли зазвичай відображаються як області з низькою щільністю пікселів.

Створимо алгоритми для обох підходів:

```
def find_letter_spacing(projx, letter_width=None, variance_threshold=5):
    """ Знаходить позиції пробілів між літерами. """
    space_positions = []

    if letter_width:
        # Шукаємо пробіли на основі ширини літери
        for i in range(len(projx)):
            if projx[i] == 0:  # Перевірка на нульові значення в проекції
                count = 0
                while i < len(projx) and projx[i] == 0:
                    count += 1
                    i += 1
                if count >= letter_width:  # Якщо ширина пробілу достатньо велика
                    space_positions.append(i - count // 2)  # Додаємо середину пробілу
```

```
    else:
        # Шукаємо пробіли на основі дисперсії
        for i in range(1, len(projx) - 1):
            # Шукаємо велику зміну щільності пікселів
            if abs(projx[i - 1] - projx[i]) > variance_threshold and
abs(projx[i + 1] - projx[i]) > variance_threshold:
                space_positions.append(i)

    return space_positions
```

Приклад використання

```
# Проекція генерується як у попередніх прикладах
projx = horizontal_projection(binary_image)

# Визначення позицій пробілів
# Для моноширинного шрифту (наприклад, ширина літери 10 пікселів)
spaces_mono = find_letter_spacing(projx, letter_width=10)

# Для шрифту з різною шириною літер
spaces_var = find_letter_spacing(projx, variance_threshold=5)

print("Позиції пробілів (моноширинний шрифт):", spaces_mono)
print("Позиції пробілів (різна ширина літер):", spaces_var)
```

Цей код спочатку шукає пробіли, використовуючи ширину літери, якщо вона відома. Якщо ширина літери не відома або якщо шрифт не є моноширинним, він шукає пробіли, використовуючи дисперсію щільності пікселів в горизонтальній проекції.

Для візуалізації позицій пробілів на оригінальному зображенні, ми можемо модифікувати функцію візуалізації, додаючи лінії на позиціях, де знаходяться пробіли. Для цього використаємо функцію cv2.line з OpenCV:

```
def visualize_spaces_on_image(binary_image, spaces, color=(255, 0,
0), thickness=1):
    """ Візуалізує позиції пробілів на бінарному зображенні
кольоровими лініями. """
    # Конвертуємо бінарне зображення у кольорове
    color_image = cv2.cvtColor(binary_image, cv2.COLOR_GRAY2BGR)

    for space in spaces:
        # Рисуємо вертикальну лінію на кожній позиції пробілу
        cv2.line(color_image,       (space,       0),       (space,
color_image.shape[0]), color, thickness)

    # Показуємо зображення
    plt.imshow(cv2.cvtColor(color_image,   cv2.COLOR_BGR2RGB))     #
Конвертуємо з BGR в RGB для вірного відображення кольорів у
Matplotlib
```

```
    plt.title('Візуалізація пробілів')
    plt.axis('off')
    plt.show()

# Використання функції
visualize_spaces_on_image(binary_image, spaces_mono)  # або
spaces_var для різної ширини літер
```

У цьому коді visualize_spaces_on_image приймає зображення та список позицій пробілів. Вона рисує вертикальну лінію на кожній з цих позицій, візуалізуючи місця, де алгоритм виявив пробіли.

У цьому коді, cv2.cvtColor(binary_image, cv2.COLOR_GRAY2BGR) перетворює бінарне зображення (відтінки сірого) у кольорове (BGR), дозволяючи нам відобразити кольорові лінії на ньому. Потім ми використовуємо cv2.line для малювання кольорових ліній на позиціях пробілів. При візуалізації за допомогою Matplotlib, ми конвертуємо зображення з BGR у RGB, оскільки Matplotlib використовує формат RGB. Можна налаштувати колір і товщину ліній за допомогою параметрів color та thickness.

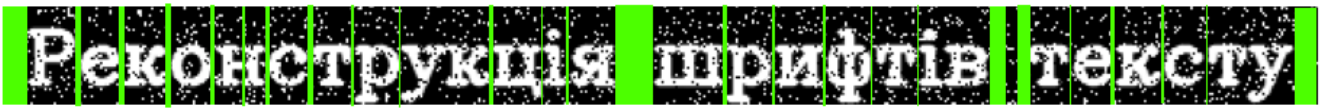


Рисунок 2.11 – Результат детекції пробілів

Як видно деякі позиції пробілів пропущені.

2.3 Алгоритм розпізнавання символів шрифту на основі шаблонів

Використання шаблонного методу для розпізнавання символів є ефективним способом. За допомогою цього методу, ми можемо визначити, чи є символ буквою чи цифрою, спираючись на його позицію в шаблоні. Після цього ми отримуємо оцінки для символу, порівнюючи його растр з еталоном. Для цифр ми отримуємо 10 оцінок, а для букв - 13. Зазвичай ми приймаємо той символ, який має найвищу оцінку.

Еталони символів зберігаються у матрицях, де чорний піксель відзначається як одиниця. Фонові пікселі представлені в комірках з відстанями до найближчого чорного пікселя, з використанням знаку мінус. Визначення відстані може виконуватися за допомогою різних формул.

$$d = \max(x, y) d = x + y,$$

де x, y – значення модулів для зсувів до найближчої чорної точки.

Розглянемо цей процес за допомогою прикладу, який можна побачити на рисунку 2.12 Для отримання оцінки ми виконуємо наступні кроки:

1. Спочатку растрове представлення символу піддається трансформації, змінюючи його розміри до відповідних розмірів еталону.

2. Отриманий трансформований символ накладається на еталон, таким чином, що відповідні значення до чорних пікселів символу підсумовуються в гніздах еталону.

3. Важливо відзначити, що велика відстань від найближчої чорної точки в еталоні значно зменшує оцінку символу при появі точки в комірці символу.

Цей процес робиться для кожного символу з використанням модулів *x* та *y* для визначення відстані до найближчої чорної точки, що допомагає нам отримати точну оцінку для розпізнання символів.

-1	-1	-1	-1	-1	-1	-2	-2
-1	1	1	1	1	-1	-1	-2
-1	1	-1	-1	-1	1	-1	-2
-1	1	-1	-1	-1	1	-1	-2
-1	1	1	1	1	-1	-1	-2
-1	1	-1	-1	-1	-1	-2	-2
-1	1	-1	-2	-2	-2	-2	-3
-1	1	-1	-2	-3	-3	-3	-3

Рисунок 2.12 - Ілюстрація еталону літери "Р".

Процес сегментації застосований не завжди досягає виділення символу за допомогою найменшого охоплюючого прямокутника. Внаслідок цього виникає необхідність коригувати положення еталону відносно растру символу. Оцінка розраховується для початкового розташування еталону на растрі і для всіх варіацій зсуву еталону відносно растру символу в восьми можливих напрямках на трохи більшу відстань - три пікселя (як показано на Рисунку 2.13).

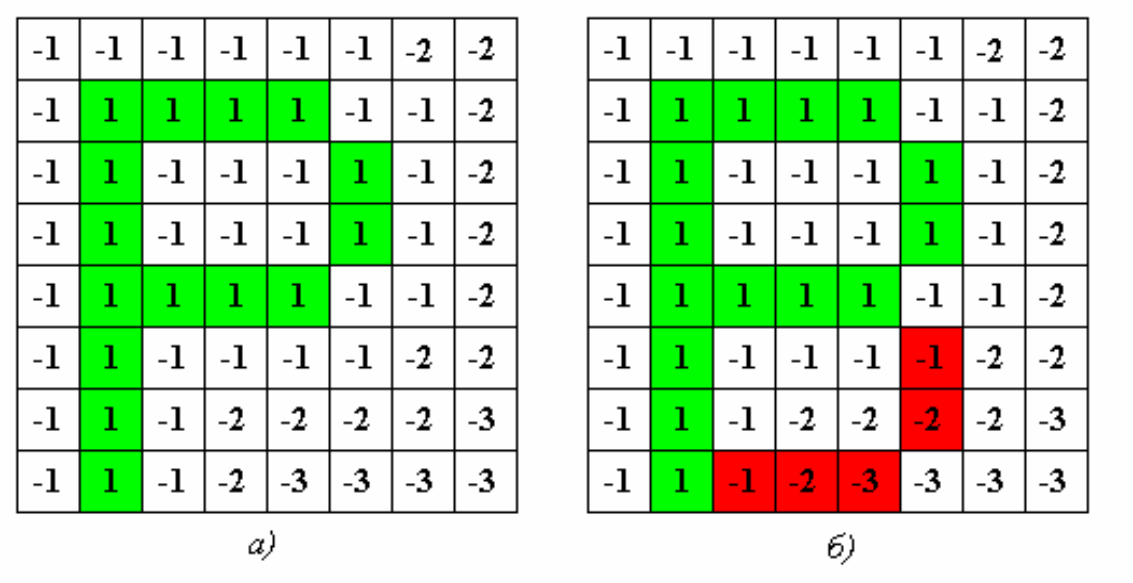


Рисунок 2.13 - Зображено

а) Оцінка для символу "Р" становить 15.

б) Оцінка для символу "В" складає 6.

На основі позиції, яка відповідає найвищій отриманій оцінці, застосовується аналогічний процес зі зсувом на один піксель. В результаті максимальний можливий зсув еталону відносно символу становить чотири пікселя, що враховує помилки у сегментації (як показано на Рисунку 2.14).

Базовий код алгоритму для розпізнавання символів:

1. Створення еталонів символів. Функція create_reference_images генерує еталони для кожної букви та цифри за допомогою cv2.putText. Еталони зберігаються у словнику reference_images.

2. Обчислення оцінки. Функція calculate_score обчислює оцінку для кожного символу, порівнюючи його з еталоном. Це просте порівняння пікселів.

3. Розпізнавання символів. Функція recognize_characters виконує розпізнавання символів на вхідному зображенні, використовуючи еталони. Вона розділяє зображення на символи (припускаючи, що символи рівномірно розподілені) і використовує оцінки для визначення найкращого варіанту.

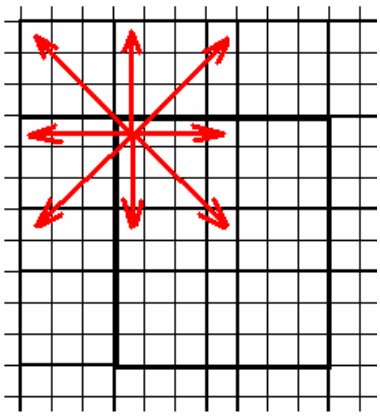


Рисунок 2.14 – Ілюстрація зсуву еталону відповідно растру символу

Цей код є прототипом і призначений для ілюстрації базового алгоритму. Для роботи з реальними зображеннями, його потрібно буде адаптувати, враховуючи різні розміри, стилі шрифтів і можливі спотворення на зображенні.

Також, зараз код містить лише визначення функцій. Для випробування вам потрібно буде завантажити тестове зображення та викликати recognize_characters із цим зображенням і словником reference_images.

2.4 Висновки до розділу

Розроблено алгоритми реконструкції на основі контурів, алгоритм виявлення символів на основі проекції, алгоритми розпізнавання символів шрифту на основі шаблонів.

3 РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Засоби розробки і структура проєкту

Вибір мови програмування та середовища розробки був обгрунтований кількома ключовими вимогами [31-42]:

1. Необхідність сумісності програми з більшістю операційних систем.

2. Доступність, низька вартість або безкоштовність середовища розробки з огляду на економічну доцільність роботи.

Ці критерії ідеально відповідають мові програмування Python та інтегрованому середовищу розробки Python IDE. Python IDE є універсальним інструментом, який уможливлює запуск програм на базі Python на різноманітних пристроях з різними операційними системами.

Python - це універсальна мова програмування високого рівня, яка зосереджена на підвищенні ефективності розробника і читаності коду. Вона характеризується лаконічним синтаксисом основи мови, одночасно пропонуючи обширну стандартну бібліотеку з широким спектром функцій.

Python підтримує кілька стилів програмування, включаючи структурне, об'єктно-орієнтоване, функціональне, імперативне та аспектно-орієнтоване програмування. Ця мова має динамічну типізацію, автоматизоване управління пам'яттю, вбудовані механізми обробки винятків, підтримує багатопотоковість і пропонує зручні високорівневі структури даних. Код Python організований у функції та класи, які можна групувати у модулі, а модулі, в свою чергу, у пакети.

Також, Python має перевагу в тому, що за допомогою спеціальних модулів можна легко запускати програми на веб-серверах, що розширює їх функціональність у декілька разів.

Для створення графічного інтерфейсу користувача (GUI) використовується бібліотека PyQT. PyQT представляє собою колекцію зв'язків з графічним фреймворком Qt для Python, який розроблений як доповнення до Python. PyQT сумісний з усіма основними платформами, що підтримують Qt, включно з Linux, іншими Unix-подібними системами, Windows і Mac OS X. У комплект PyQT входить також інструмент Qt Designer (Qt Creator) для розробки GUI. За

допомогою утиліти pyuic можна перетворювати файли, створені в Qt Designer, у код Python, що робить PyQT ефективним інструментом для розробки прототипів. Додатково, PyQT дозволяє інтегрувати нові графічні контрольні елементи, розроблені на Python, в Qt Designer.

Бібліотека PyQT є надзвичайно потужним інструментом, що майже повністю реалізує всі можливості бібліотеки Qt, представляючи понад 600 класів функціональності, включаючи:

1. Розширений набір віджетів GUI, що дозволяє створювати і налаштовувати інтерфейс користувача з великою кількістю можливостей.

2. Високопродуктивні стилі віджетів, які допомагають надавати інтерфейсу сучасний і привабливий вигляд.

3. Можливість легко взаємодіяти з різними базами даних, включаючи Mysql, ODBC, PostgreSQL, Oracle, завдяки вбудованій підтримці SQL.

4. Підтримка інтернаціоналізації (i18n), що дозволяє легко адаптувати програму для різних мов і регіональних налаштувань.

5. Вбудований XML синтаксичний аналізатор для роботи з XML-даними.

6. Підтримка SVG для відображення векторної графіки.

7. Можливість відтворення відео та аудіо, що відкриває безліч можливостей для розробки мультимедійних програм.

Для створення і налаштування інтерфейсу користувача використовується Qt Designer - міжплатформний компонувальник, який дозволяє швидко створювати віджети та діалогові вікна з використанням тих самих віджетів, які будуть використовуватися у програмі. Форми, створені за допомогою Qt Designer, є повністю функціональними і можуть бути переглянуті в режимі реального часу.

Для візуалізації даних та побудови графіків використовується бібліотека графіки Matplotlib. Matplotlib розповсюджується під ліцензією BSD і надає безліч можливостей для створення різних типів графіків і таблиць, включаючи графіки, діаграми розсіювання, гістограми, кругові графіки, стовпчасті графіки, контурні графіки, градієнтні поля, спектральні графіки та багато інших. Використання Matplotlib разом із NumPY, SciPY та IPython дозволяє створювати високоякісні

графіки та візуалізації даних, надаючи можливості, подібні до MATLAB, але з відкритим вихідним кодом.

Можливості використання мови програмування Python дуже різноманітні та дозволяють користувачам розкрити свою творчість та досягти бажаних результатів у багатьох областях:

1. Графічний інтерфейс**:** Можливість вказувати осі координат, робити сітки, додавати мітки та пояснення, а також використовувати логарифмічний масштаб або полярні координати робить Python ідеальним інструментом для створення інтерактивних та інформативних графічних інтерфейсів.

2. 3D-графіка**:** Можливість створювати 3D-графіку дозволяє візуалізувати дані у тривимірному просторі, що відкриває безліч можливостей для відображення складних моделей та даних.

3. Картографія**:** Python може бути використаний для створення географічних додатків, а також аналізу та відображення географічних даних.

4. Робота з Excel: Python може легко інтегруватися з Excel для автоматизації обробки та аналізу даних, що зберігаються у таблицях Excel.

5. GTK та інше**:** Використання різних бібліотек та інструментів, таких як GTK, дозволяє створювати різноманітні програми для графічного інтерфейсу користувача та інші застосування.

6. Анімація**:** Matplotlib може бути використаний для створення анімованих зображень, що відкриває можливості для створення динамічних інформаційних візуалізацій.

7. Обробка зображень**:** Бібліотека обробки зображень Python PIL надає широкий спектр можливостей для завантаження, перекладу та обробки зображень різних форматів, включаючи підтримку різних колірних просторів, зміну форматів зображень та застосування фільтрів.

8. Інтеграція з PyQT і Tkinter**:** Python дозволяє легко інтегрувати ці інструменти для створення більш складних інтерфейсів та додатків.

Загалом, мова програмування Python є потужним інструментом для творчих завдань у багатьох галузях, надаючи доступ до багатьох бібліотек та інструментів для досягнення різноманітних цілей.

Існує безліч причин, які роблять мову програмування Python відмінним вибором [43-52] для виконання різних завдань:

1. Платформонезалежність**:** Python не обмежений жодною конкретною операційною системою або машиною, що дає можливість запускати програми на різних платформах без значних зусиль.

2. Підтримка різних парадигм програмування: Python підтримує основні парадигми програмування, такі як об'єктно-орієнтоване, функціональне та процедурне програмування. Це дозволяє вибирати підходящий стиль програмування для конкретної задачі.

3. Зручний синтаксис: Код на Python легко читається і зрозумілий для розробників, що полегшує співпрацю та підтримку програм.

4. Багатий екосистема бібліотек**:** Python має велику кількість корисних бібліотек, які спрощують розробку та дозволяють виконувати різні завдання без перевантаження коду.

5. Бібліотека Numpy**:** Вона розширює можливості Python, надаючи підтримку великих багатомірних масивів та матриць, а також високорівневих математичних функцій. Все це працює дуже швидко завдяки використанню вставок мов C, C++ і Fortran.

6. OpenCV для обробки зображень**:** Бібліотека алгоритмів комп'ютерного зору OpenCV, з відкритим вихідним кодом, дозволяє легко працювати з обробкою зображень та чисельними алгоритмами загального призначення.

7. Підтримка роботи з зображеннями**:** Обробка зображень є невід'ємною частиною багатьох завдань, і Python забезпечує зручні засоби для її виконання. Кількість кольорів точок зображення та їх стани важливі для досягнення високої продуктивності роботи.

Загалом, Python стає досконалим інструментом для виконання різноманітних завдань завдяки своїм властивостям і багатому екосистемі

бібліотек. Ця мова дозволяє програмістам розкрити свій потенціал і досягти бажаних результатів у різних областях роботи.

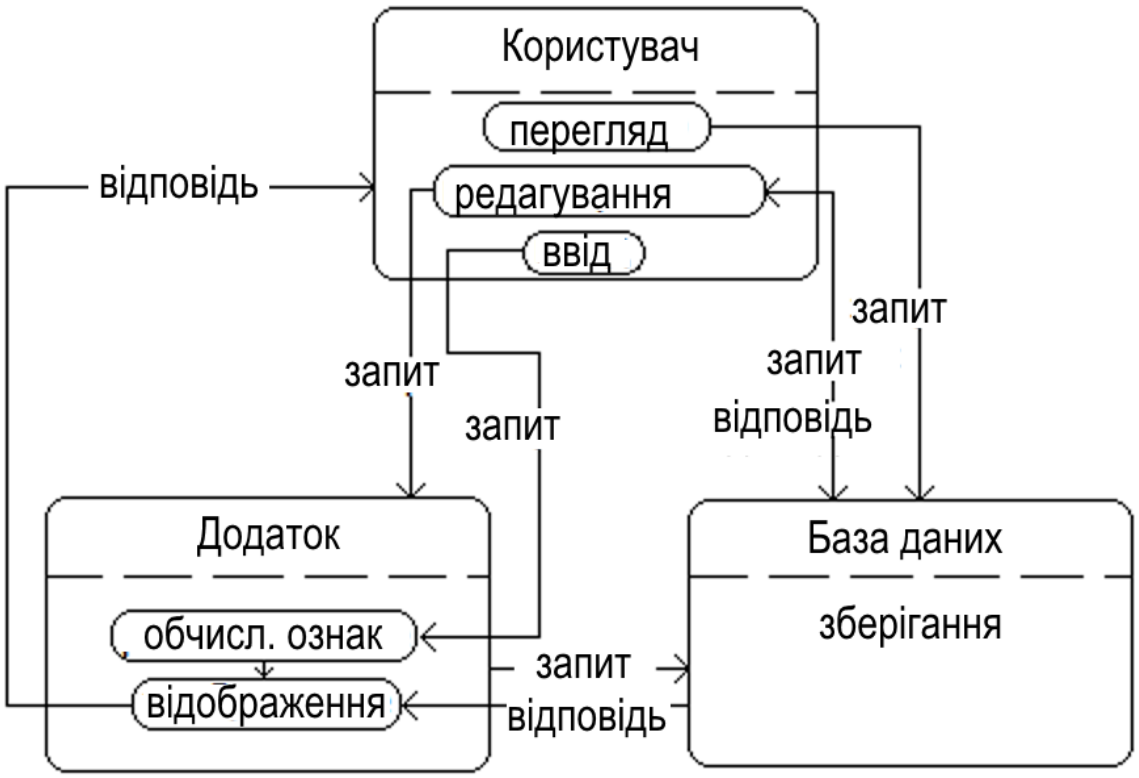


Рисунок 3.1 - Функціональна схема системи розпізнавання

При створенні програмного продукту, однією з основних настанов є забезпечення відповідності функціональності продукту вимогам та очікуванням користувачів. У цьому розділі ми зануримося в деталі функціональної архітектури програмного забезпечення, яке ми розробляємо у рамках даної дипломної роботи.

Головною метою нашого завдання є розробка модуля обчислення ознак, який є невід'ємною частиною нашої програми. Мінімальний функціональний набір включає в себе зберігання, відображення та пошук інформації. Розглянемо класифікацію цієї функціональності та можливі варіанти її подальшого розвитку, як це показано на рисунку 3.2.

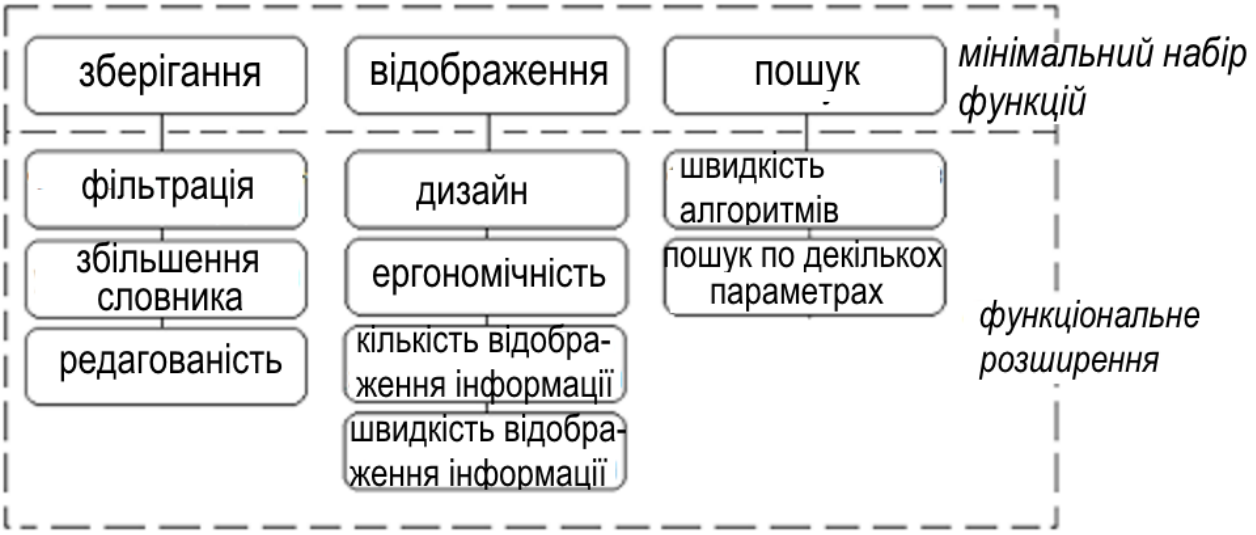


Рисунок 3.2 - Особливості Функціонування ПЗ

Основна мета при створенні цього програмно комплексу полягала не в його багатофункціональності, а в аналізі та впровадженні методів розпізнавання образів. Проте, комплекс має значний спектр функцій, і велика увага приділялася не лише методам розпізнавання, а й якості візуалізації та збереження даних.

Ось перелік ключових особливостей програми:

Розширені можливості для вимірювання відстаней: інтерфейс для додавання нових метрик, що дозволяє користувачам включати різноманітні типи зображень, збагачуючи функціонал системи.

Вдосконалення процесу розпізнавання зображень: перед розпізнаванням кожне зображення проходить детальну обробку, включаючи видалення шумів, що покращує якість та швидкість обробки.

Гнучкий дизайн і ергономіка інтерфейсу: враховуючи різноманітність користувацьких потреб, візуальне виконання можна налаштовувати, не обмежуючись конкретним стилем, із залежністю від налаштувань операційної системи користувача.

Ефективне відображення інформації: програма забезпечує швидкий доступ до необхідної інформації, пропонуючи зручний пошук за декількома критеріями, і виконана в стандартному стилі, що відповідає початковим версіям комерційних продуктів чи версіям для тестування в реальних умовах.

Оптимізація дизайну: дизайн програми може бути змінений після підтвердження надійності роботи модуля розпізнавання.

Заключна версія програми в цій дипломній роботі містить описані вище функції і відповідає вимогам до дипломного проекту, при цьому передбачається подальше розширення та доповнення комплексу.

Визначення вибору інструментів для створення програмного рішення. Підбір мови програмування та середовища розробки здійснювався на основі декількох ключових критеріїв:

- Необхідність сумісності програми з більшістю операційних систем;

- Важливість доступності і економічної вигідності використовуваного середовища розробки, виходячи з міркувань бюджету проекту;

- Підтримка мобільних платформ, таких як Windows Mobile та Android OS, щоб забезпечити можливість запуску програмно-математичного комплексу на портативних пристроях, розширюючи сферу його застосування;

- Перевага надається реалізації системи у форматі веб-застосування, що в певних випадках сприяє уникненню необхідності встановлення програми на комп'ютер, оптимізує процес оновлення баз даних, дозволяє управляти та адаптувати інтерфейс програми в реальному часі.

3.2 Об’єктна модель та інтерфейс

Для створення об'єктної моделі програмного забезпечення за функціональними вимогами (див. 1.5), можна використати наступні класи:

Клас OCRModule відповідає за виявлення та вибір літер шрифту з зображення.

- detect_text(image) виявляє та повертає текст із зображення.

- extract_font_letters(text, font_specification) виділяє літери, які відповідають заданим специфікаціям шрифту.

Клас VectorizationTool для векторизації растрових зображень літер.

- raster_to_vector(image) перетворює растрове зображення літери на векторний формат.

- ensure_accuracy(vector_image) перевіряє та покращує точність векторизації.

Клас FontGenerator відповідає за створення TTF файлу шрифту.

- generate_ttf(vector_letters): Генерує TTF файл шрифту на основі векторизованих літер.

- validate_ttf_standards(ttf_font): Перевіряє TTF файл на відповідність стандартам.

Клас TextGenerator для генерації тексту з використанням створеного шрифту.

- generate_text(image) генерує текст із зображення за допомогою OCR, використовуючи власний шрифт.

- correct_errors(text) виправляє помилки, виявлені в процесі генерації тексту.

Початкова діаграма класів наведена на рисунку 3.3

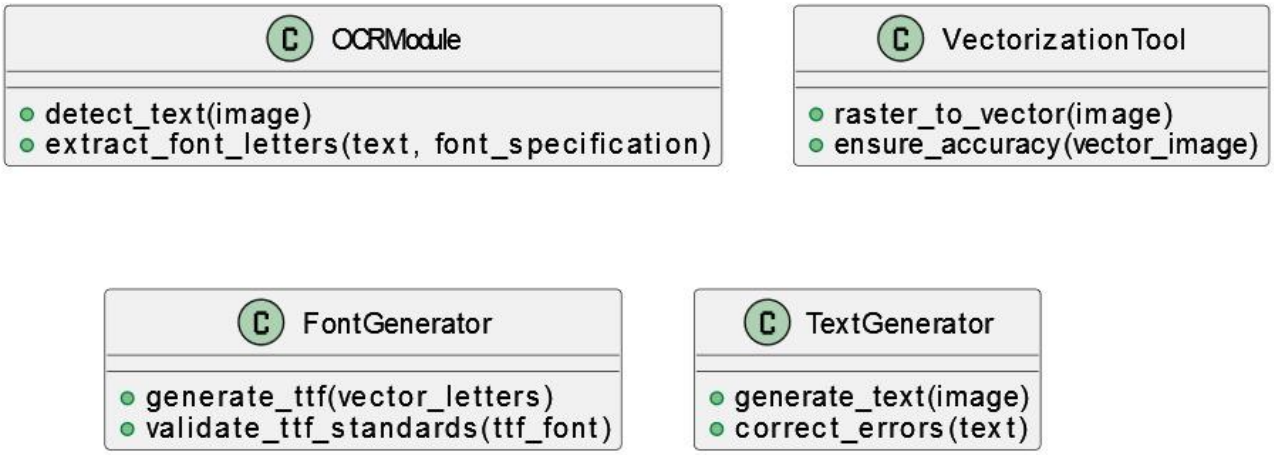


Рисунок 3.3 – Об’єктна модель програмного забезпечення

Основний алгоритмом реконструкції є алгоритм на основі аналізу шаблонів.

```
def create_reference_images():
    """
    Create reference images for each character using cv2.putText.
    """
    chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
    reference_images = {}

    for char in chars:
```

```
        img = np.zeros((40, 40), dtype=np.uint8)
        cv2.putText(img, char, (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
1, (255), 2, cv2.LINE_AA)
        reference_images[char] = img

    return reference_images
```

Функція оцінки схожості між еталоном та знайденим об’єктом кандидатом.

```
def calculate_score(img, ref_img):
    """
    Calculate the score of an image against a reference image.
    """
    score = np.sum(img == ref_img)
    return score
```

Функція розпізнавання символу.

```
def recognize_characters(image, reference_images):
    recognized_text = ""
    character_width = image.shape[1] // 5
    for i in range(5):
        character_image = image[:, i * character_width:(i + 1) *
character_width]
        best_match = None
        highest_score = 0
        for char, ref_img in reference_images.items():
            score = calculate_score(character_image, ref_img)
            if score > highest_score:
                highest_score = score
                best_match = char
        recognized_text += best_match if best_match else '?'
    return recognized_text
```

Діаграму активності UML зображено на рисунку 3.3.

Створення візуального інтерфейсу програми. Необхідно, щоб програма мала основне вікно для демонстрації вхідного зображення, області для показу текстурних характеристик та сегментованого зображення. Для візуалізації сегментованого зображення слід використовувати псевдоколірну обробку або контури областей, виділені іншим кольором. Таким чином, основне вікно має включати зону для вхідного зображення, кнопки для перемикання між режимами відображення, а також поля для введення параметрів алгоритму сегментації. Презентуємо знімок екрану основного вікна програми з позначеннями важливих елементів та класів віджетів на рисунку 3.4.

Рисунок 3.3 - UML діаграма активності

Клас Q MenuBar забезпечує функціональність меню на верхньому рівні, яке є інтегральною складовою більшості програмних додатків. Розташоване у верхній частині головного вікна, це меню служить місцем для вміщення численних команд, серед яких користувач може обирати потрібну. Традиційно в програмах використовуються чотири основні типи меню:

- верхнього рівня;
- спливаюче;
- від'ємне, яке може бути відокремлене від головного;
- контекстне.

У бібліотеці Qt реалізація меню здійснюється за допомогою класу Q Menu. Головна функція цього класу полягає у розміщенні команд всередині меню. Кожна команда є об'єктом дії, а всі дії та саме меню можуть бути пов'язані зі слотами для активації відповідного коду при виборі користувачем певної команди.

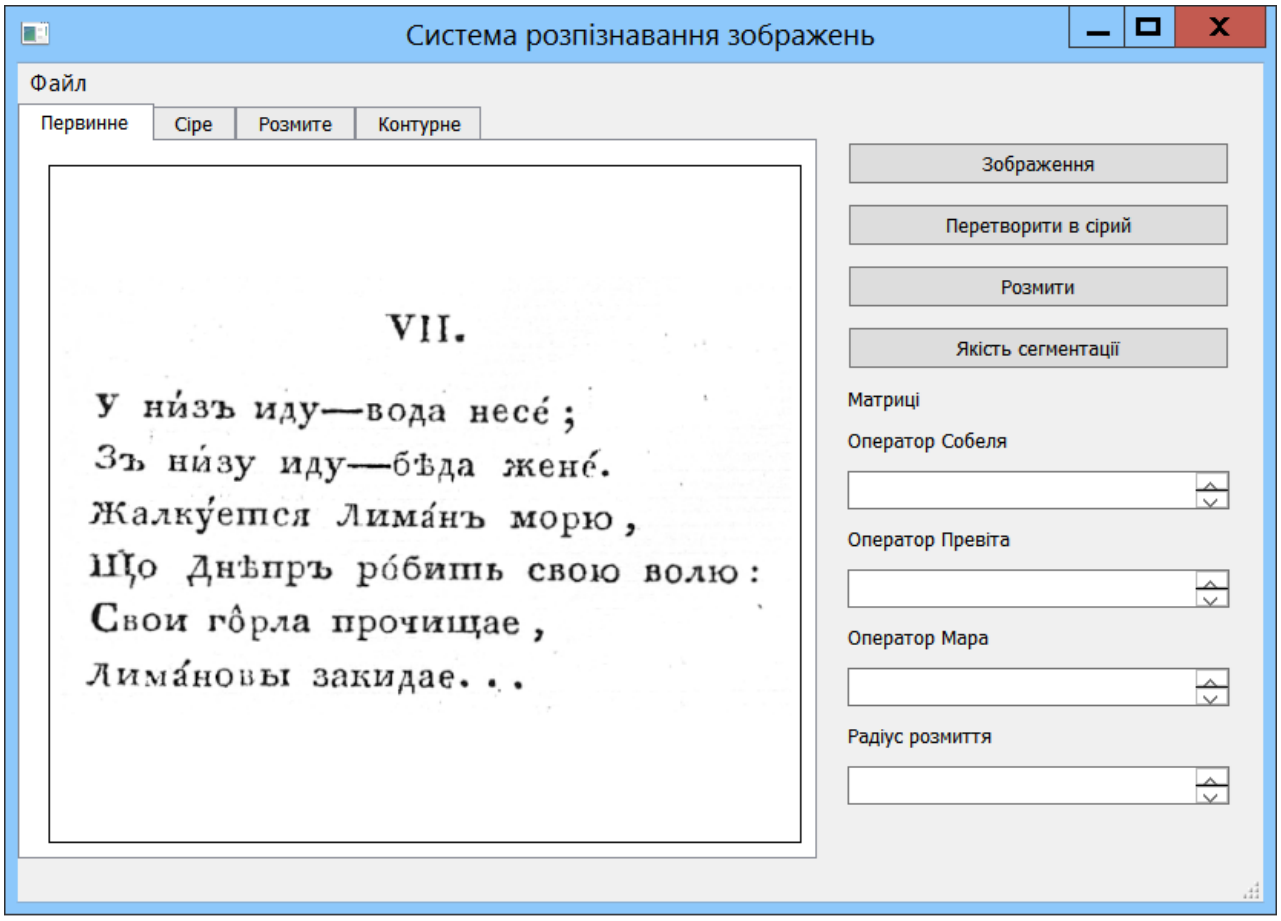


Рисунок 3.4 – Зразок інтерфейсу програми

Ключовим елементом у меню є головне меню, відоме як меню верхнього рівня. Це стало відображене меню містить набір команд, які можна активувати за допомогою миші або клавіатурних скорочень (<Alt> разом із клавішами курсору). Основні команди цього меню використовуються для виведення спадних підменю.

Клас QTabWidget представляє собою комплекс віджетів для вкладок. Віджет вкладок складається з панелі вкладок та "області сторінок" для демонстрації сторінок, асоційованих з кожною вкладкою. Зазвичай панель вкладок розташована вгорі області сторінок, але існують різноманітні варіанти конфігурації. Кожна вкладка асоційована з окремим віджетом, відомим як сторінка. У області сторінок видимою є тільки активна сторінка, усі інші залишаються прихованими. Користувач може переглянути сторінку, натиснувши на відповідну вкладку або використавши гарячі клавіші.

Для роботи з QTabWidget необхідно виконати наступні кроки:

1. Створити екземпляр QTabWidget;
2. Створити окремий QWidget для кожної сторінки вкладок, без встановлення батьківського віджета;
3. Додати дочірні віджети на сторінковий віджет, використовуючи компонувальники для розміщення;
4. Використати методи addTab() або insertTab() для додавання віджетів сторінок на віджет вкладок, присвоюючи кожній вкладці відповідну мітку та гарячі клавіші.

При виборі сторінки користувачем відправляється сигнал currentChanged(). Індекс активної сторінки можна дізнатися через currentIndex(), а віджет активної сторінки - через currentWidget().

Віджет QTableWidget функціонує як таблиця, яка представляє собою двовимірний решетчатий масив. Цей віджет дозволяє переглядати частину таблиці, користувач може навігувати по ній, скориставшись прокруткою. Коли користувач вводить текст в пусту комірку, QTableWidget автоматично створює об'єкт QTableWidgetItem для зберігання цього тексту. QTableWidget включає декілька вкладених віджетів: горизонтальний заголовок QHeaderView зверху,

вертикальний заголовок QHeaderView зліва та два слайдери QScrollBar. В центральній частині розташований спеціальний віджет, названий Viewport, де QTableWidget відображає комірки. Різні вкладені віджети доступні через функції, успадковані від QTableView та QAbstractScrollArea. QAbstractScrollArea має рухому область перегляду та два слайдери, які можна включати або вимикати.

Віджет QPushButton служить як кнопка команди у графічному інтерфейсі користувача, є одним із найбільш поширених елементів керування. Натискання кнопки сигналізує про бажання користувача виконати певну дію або відповісти на запитання. Типові приклади кнопок включають такі, як OK, Застосувати, Скасувати, Закрити, Так, Ні, Довідка. Вони зазвичай містять текстову мітку і за потреби маленьку іконку.

Релізуємо додатковий компонент системи для пошуку зображень шрифтів по змісту. Існують три категорії методів пошуку зображень: базований на метаданих, заснований на візуальному зразку, та комбінований метод, який інтегрує обидва зазначені підходи.

Метод, заснований на метаданих, мало чим відрізняється від звичайних текстових пошукових систем, які використовують ключові слова. Цей підхід спирається на описові дані, які не обов'язково відображають зміст самого зображення. Замість цього, пошук базується на текстовій інформації, такій як (1) вручну створені анотації та теги, та (2) контекстні дані, автоматично згенеровані з тексту, розташованого поруч із зображенням на веб-сайті. Пошукові запити в таких системах мало чим відрізняються від звичних запитів у текстових пошукових системах, і результатом є зображення, пов'язані з аналогічними метаданими.

З іншого боку, методи пошуку за зразком працюють виключно з візуальним змістом зображення, ігноруючи текстові описи. У такому підході зображення піддаються аналізу, кількісній оцінці та каталогізації на основі їх візуальних характеристик, щоб при пошуку можна було ідентифікувати та повертати подібні зображення. Термін "системи пошуку за зразком" часто зустрічається в академічних джерелах, проте, по суті, це просто один із способів пошуку визначення "системи пошуку зображень", який підкреслює, що система

орієнтується виключно на візуальні аспекти зображень без врахування текстових описів або анотацій.

У системах візуального пошуку впроваджений спеціалізований алгоритм, який вилучає характеристичні вектори — це набір числових величин, які слугують для абстрактного кількісного представлення образу. Коли з'являється запит у вигляді зображення, з нього екстрагуються ці характеристики, які наступно порівнюються з базою даних характеристик для ідентифікації аналогічних зображень. Такі системи пошуку операційно залежать від візуального вмісту зображення. Вони можуть бути складними в реалізації та розширенні, проте мають перевагу у формі повністю автономних алгоритмів для управління пошуком, що виключає потребу в ручному втручанні.

Для запуску системи візуального пошуку вимагається індексація даних. Індексація — це процедура оцінювання і аналізу зображень за допомогою дескриптора, який вилучає характеристики кожного зображення. Дескриптори зображення можуть використовувати різні алгоритми для опису характеристик, включаючи:

- Середні значення та стандартні відхилення для кольорових каналів;
- Статистичні моменти для опису форми образу;
- Величину градієнту та орієнтацію для деталізації текстури та форми.

Таким чином, дескриптори визначають методику оцінювання образів, а характеристичні вектори, що є результатом цього процесу, представляють зображення у вигляді числового набору для подальшого аналізу та порівняння (див. рисунок 3.5).

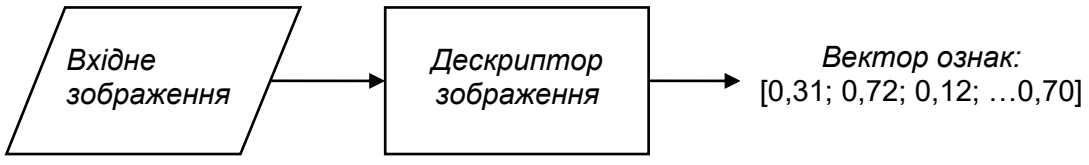


Рисунок 3.5 –Дескриптор зображення шрифта

Для аналізу відповідності між векторами характеристик зазвичай вдаються до застосування метрик або алгоритмів визначення схожості. Такі інструменти

оцінки приймають пару векторів характеристик як аргументи та видають числове значення, яке відображає рівень їхньої схожості. Процедура порівняння двох зображень, базуючись на їх векторах характеристик, демонструється на рисунку 3.6.

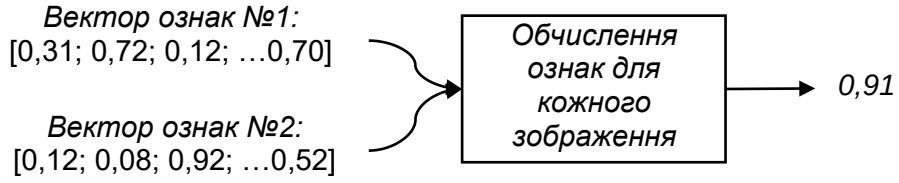


Рисунок 3.6 – Процес порівняння між двома образами

Порівняння двох образів відбувається через передачу їх векторів характеристик у спеціалізовану функцію, яка вимірює відстань чи схожість. Випливаюче з цього, числове значення служить як індикатор ступеню подібності між образами. Це число, зазвичай дробове, слугує міркою для оцінювання взаємної схожості образів.

Чотирьохетапний алгоритм розробки систем візуального пошуку зображень включає:

1. Вибір описів зображень. На цій стадії визначаються ключові властивості образу для аналізу, такі як колірні характеристики та структурні елементи.

2. Індексація бази даних. Застосування обраного опису до кожного образу в базі даних і збереження обчислених характеристик у відповідному сховищі для наступного порівняння.

3. Розробка метрики схожості. Створення алгоритмів оцінки подібності, з використанням таких підходів як евклідова відстань, косинусна відстань чи Хі-квадрат відстань, вибір яких залежить від специфіки даних і характеристик образу.

4. Виконання пошуку. На цьому етапі користувач запитує образ через інтерфейс системи, яка, у свою чергу, обчислює характеристики для запиту та порівнює їх з індексованими даними, повертаючи найбільш відповідні образи.

Ці кроки являють собою основу для створення системи візуального пошуку. Зі збільшенням складності систем і різноманіттям використовуваних

характеристик, кількість процедур може розширюватися, вимагаючи додаткових дій для кожного з вищезазначених етапів. Рисунок 3.7 демонструє перші два кроки цього процесу.

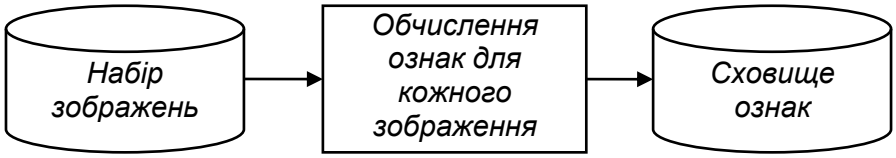
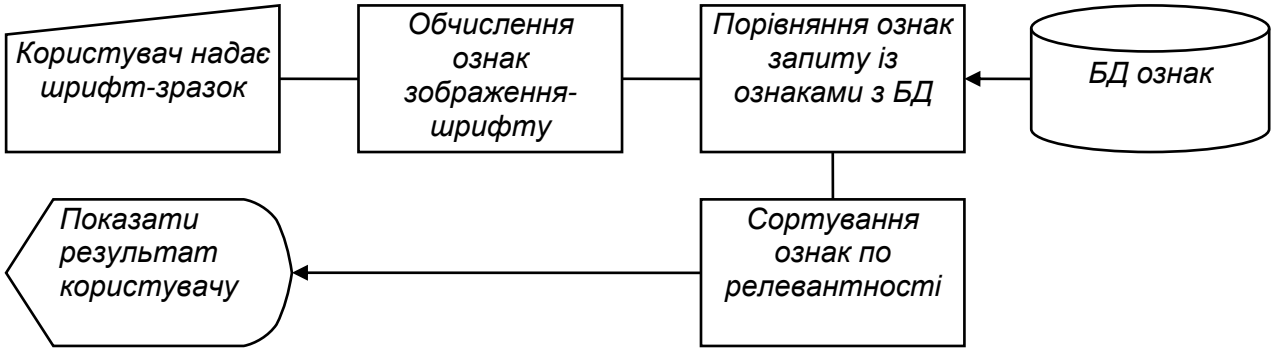


Рисунок 3.7 – Графічне зображення етапу екстракції характеристик з образів у колекції даних.

Процедура починається з витягування визначальних характеристик з кожного образу в колекції. Наступним кроком є занесення цих характеристик до репозиторію даних. Після цього (рисунок 3.8) відбувається перехід до модулю пошуку, який охоплює третій та четвертий кроки процесу.



Ілюстрація 3.8 – Діаграма алгоритму пошуку у візуальній пошуковій системі

Коли користувач ініціює запит за допомогою обраного образу, система генерує опис цього запиту шляхом вилучення векторів характеристик. Ці вектори потім зіставляються з характеристиками, що вже є у базі даних, з метою виявлення найбільш схожих образів. Результати ранжуються відповідно до ступеня схожості та надаються користувачу для огляду.

3.3 Експериментальне дослідження алгоритмів

Ціль експерименту для аналізу точності розпізнавання**:** визначити точність OCR-системи при конвертації зображень тексту в редагований текстовий формат. Об'єкти дослідження:

- Різноманітні зображення тексту, що включають друковані та рукописні документи.

- OCR-система, яка буде використана для розпізнавання тексту.

Методика:

1. Підготовка даних. Вибрати зразки зображень тексту, що включають різні шрифти, розміри шрифтів, якості зображень (чіткі та нечіткі), та різні стилі написання (друковані та рукописні).

2. Розпізнавання тексту. Використати OCR-систему для перетворення зображень тексту в редагований текстовий формат.

3. Оцінка точності. Порівняти розпізнаний текст з оригінальним текстом зображення, використовуючи метрики, такі як точність на рівні символів, слів та фраз.

Критерії оцінювання:

- Точність на рівні символів відсоток правильно розпізнаних символів.

- Точність на рівні слів відсоток правильно розпізнаних слів.

- Точність на рівні фраз відсоток правильно розпізнаних фраз або речень.

Аналіз даних. Здійснити статистичний аналіз отриманих даних для визначення загальної ефективності та точності OCR-системи. Включити аналіз помилок для ідентифікації частих проблем, таких як нерозпізнавання окремих символів або погане розпізнавання при певних умовах освітлення чи якості зображення.

Публічні датасети є надзвичайно важливими для розвитку та тестування систем оптичного розпізнавання символів. Вони дозволяють розробникам та дослідникам оцінювати ефективність та точність своїх систем у реальних умовах. Ось декілька відомих публічних датасетів для OCR:

1. Датасет IAM Handwriting Database [31] включає велику кількість зразків рукописного тексту, який можна використовувати для тренування та тестування OCR-систем, що фокусуються на розпізнаванні рукописного тексту.

2. Датасет The Street View Text (SVT) Dataset [30] складається з фотографій вивісок, вивісок магазинів та іншого тексту, знятого у міському середовищі. Він є корисним для розробки OCR-систем, які можуть розпізнавати текст у різних умовах освітлення та з різними фонами.

3. Французький датасет Rimes Dataset містить зразки рукописного тексту та підходить для вивчення та оцінки OCR-систем, які спеціалізуються на розпізнаванні рукописного тексту.

4. Історичний датасет «The George Washington Papers» містить зразки письма Георга Вашингтона. Він є корисним для розробки OCR-систем, здатних розпізнавати історичні документи.

5. Датасет «The Chars74K Dataset» містить зображення цифр та літер, взятих з різних джерел. Це включає як друковані, так і рукописні символи, і є корисним для розробки та тестування OCR-систем, які фокусуються на символьному рівні.

6. Датасет «MNIST Database» в основному використовується для розпізнавання рукописних цифр, він також може бути використаний для тестування OCR-систем, оскільки він містить велику кількість зразків рукописних цифр.

7. European Parliament Proceedings Parallel Corpus [32] Цей багатомовний датасет містить офіційні записи процедур Європейського парламенту. Він корисний для OCR-систем, орієнтованих на роботу з багатомовними документами.

Ці датасети дозволяють оцінити різні аспекти OCR-систем, включаючи точність розпізнавання, спроможність працювати з різними шрифтами та стилями написання, а також ефективність у різних умовах (наприклад, при різному освітленні чи якості зображення).

Проведено експериментальне дослідження розпізнавання тексту при різних рівнях спотворення. Результат зображено на рисунку 3.9.

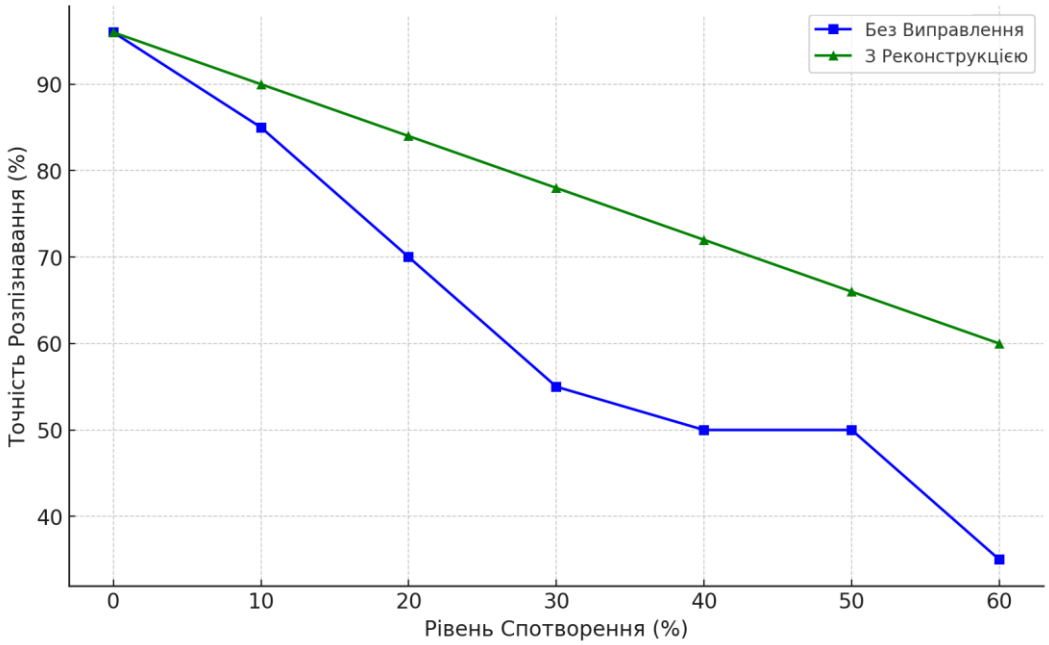


Рисунок 3.9 – Графік залежності точності розпізнавання від рівня спотворень

Експерментальне дослідження показало значну первагу в точності. В залежності від рівня спотворень перевага склала від 25% до 5% .

3.4 Висновки до роздліу

Розроблено архітектуру програмних засобів. Для реалізації обрано мову прогмування Python, бібліотеки OpenCv, scikit-learn, scikit-image та ін. Програмно реалізовано алгоритми реконструкції шрифтів та компоненти системи оптичного розпізнавання символів на основі співставлення із шаблонами. Проведено експериментальне дослідження розроблених алгоритмів при різних рівнях спотворення. Результати показали що в залежності від рівня спотворень підвищення точності склаладає від 25% до 5% .

ВИСНОВКИ

1. Проведено аналітичний огляд існуючих алгоритмів аналізу шрифтів. Систематизовано види спотворень символів шрифтів у стрічках тексту. Вони класифікуються, залежно від джерела (сканування, друк, рукопис), типу (геометричні, фотографічні, текстові) або впливу на текст (розмиття, злипання, відсутність елементів). Сформовано технічне завдання на розроблення програмного забезпечення.

2. Розроблено алгоритми реконструкції шрифтів на основі обчислення топологічних характеристик ключових точок скелетного представлення контуру, що дозволило з'єднувати фрагменти символу шрифту.

3. Розроблено алгоритм виявлення символів на основі проекції, що дозволило виявляти пробіли між символами шрифту в стрічці тексту.

4. Розроблено алгоритм реконструкції шрифту на основі шаблонів, що дозволило відновлювати невпевнено розпізнані символи і підняти точність розпізнавання.

5. Розроблено архітектуру програмного засобу оптичного розпізнавання тексту. Для реалізації обрано мову програмування Python, бібліотеки OpenCv, scikit-learn, scikit-image та ін. Програмно реалізовано алгоритми реконструкції шрифтів та компоненти системи оптичного розпізнавання символів на основі співставлення із шаблонами. Проведено експериментальне дослідження розроблених алгоритмів при різних рівнях спотворення. Результати показали що в залежності від рівня спотворень підвищення точності розпізнавання складає від 25% до 5%.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Винницька Т.П., Костенюк А.В. Аналіз структури тексту при оптичному розпізнаванні документів. Матеріали науково-практичної конференції молодих вчених і студентів «Інтелектуальні комп'ютерні системи та мережі» (ІКСМ-2023)., м. Тернопіль, 5 груд. 2023 р. С. 24.

2. Костенюк А.В., Винницька Т.П. Реконструкція шрифтів при оптичному розпізнаванні тексту. Матеріали науково-практичної конференції молодих вчених і студентів «Інтелектуальні комп'ютерні системи та мережі» (ІКСМ-2023)., м. Тернопіль, 5 груд. 2023 р. С. 49.

3. Березький О.М., Дубчак Л.О., Мельник Г.М. Методичні рекомендації до виконання кваліфікаційної роботи освітнього ступеня «Магістр». Спеціальність: комп'ютерна інженерія. Магістерська програма – «Комп'ютерна інженерія» / Під ред. О.М. Березького. Тернопіль: ЗУНУ, 2020. 32 с.

4. 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.

5. Zhao N., Cao Y., Lau R. Modeling fonts in context: Font prediction on web designs. *Computer Graphics Forum*. 2018. P. 385--395.

6. Synthtiger: Synthetic text image generator towards better text recognition models / M. Yim et al. *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR)*. 2021. P. 109--124.

7. Deepfont: Identify your font from an image / Z. Wang et al. *Proceedings of the International Conference on Multimedia*. 2015. P. 451--459.

8. Tibshirani R., Walther G., Hastie T. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*. 2001. Vol. 63, no. 2. P. 411--423.

9. Takeshita K., Shioyama J., Uchida S. Label or message: A large-scale experimental survey of texts and objects co-occurrence. *Proceedings of the International Conference on Pattern Recognition (ICPR)*. 2021. P. 6227--6234.

10. Dropout: a simple way to prevent neural networks from overfitting / N. Srivastava et al. *The Journal of Machine Learning Research*. 2014. Vol. 15, no. 1. P. 1929--1958.

11. Serif or sans: Visual font analytics on book covers and online advertisements / Y. Shinahara et al. *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR)*. 2019. P. 1041--1046.

12. Exploratory font selection using crowdsourced attributes / P. O'Donovan et al. *ACM Transactions on Graphics*. 2014. Vol. 33, no. 4. P. 1--9.

13. Distributed representations of words and phrases and their compositionality / T. Mikolov et al. *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*. 2013.

14. Efficient estimation of word representations in vector space / T. Mikolov et al. *International Conference on Learning Representations*. 2013.

15. Matsumura S., Choi S., Aizawa K. Font search across various languages based on multimodal learning. *Proceedings of the Conference on Multimedia Information Processing and Retrieval (MIPR)*. 2020. P. 173--176.

16. Kulahcioglu T., Melo G. D. Fonts like this but happier: A new way to discover fonts. *Proceedings of the International Conference on Multimedia*. 2020. P. 2973--2981.

17. Kulahcioglu T., Melo G. D. Paralinguistic recommendations for affective word clouds. *Proceedings of the International Conference on Intelligent User Interfaces*. 2019. P. 132--143.

18. Kulahcioglu T., Melo G. D. Predicting semantic signatures of fonts. *Proceedings of the International Conference on Semantic Computing (ICSC)*. 2018. P. 115--122.

19. Openimages: A public dataset for large-scale multi-label and multi-class image classification / I. Krasin et al. 2017.

20. Kingma D., Ba J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. 2014.

21. Visual font pairing / S. Jiang et al. *IEEE Transactions on Multimedia*. 2019. Vol. 22, no. 8. P. 2086--2097.

22. Ikoma M., Iwana B., Uchida S. Effect of text color on word embeddings. *Proceedings of the International Workshop on Document Analysis Systems (DAS)*. 2020. P. 341--355.

23. Deep residual learning for image recognition / K. He et al. *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016. P. 770--778.

24. BERT: pre-training of deep bidirectional transformers for language understanding / J. Devlin et al. *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2019. P. 4171--4186.

25. Choi S., Matsumura S., Aizawa K. Assist users' interactions in font search with unexpected but useful concepts generated by multimodal learning. *Proceedings of the International Conference on Multimedia Retrieval*. 2019. P. 235--243.

26. Large-scale visual font recognition / G. Chen et al. *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014. P. 3598--3605.

27. Character region awareness for text detection / Y. Baek et al. *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019. P. 9365--9374.

28. Analyzing Font Style Usage and Contextual Factors in Real Images / N. Yasukochi et al. *Document Analysis and Recognition - ICDAR 2023 - 17th International Conference, San Jose, CA, USA, August 21-26, 2023, Proceedings, Part III*. 2023. P. 331--347. URL: 10.1007/978-3-031-41682-8_21.

29. Sundermeyer M., Schlüter R., Ney H. LSTM neural networks for language modeling. *Proc. Interspeech 2012*. 2012. P. 194--197. URL: 10.21437/Interspeech.2012-65.

30. Papers with Code - SVT Dataset. The latest in Machine Learning | Papers With Code. URL: https://paperswithcode.com/dataset/svt.

31. IAM Handwriting Database. Research Group on Computer Vision and Artificial Intelligence – Computer Vision and Artificial Intelligence. URL: https://fki.tic.heia-fr.ch/databases/iam-handwriting-database.

32. Europarl Parallel Corpus. Statistical and Neural Machine Translation. URL: https://www.statmt.org/europarl/.

33. The Analysis of Performance for Priority Distinction Double-queue and Double-server Communication Network / J. L. Xiong et al. *International Journal of Communication*. 2014. Vol. 3. URL: https://api.semanticscholar.org/CorpusID:62382654.

34. Du J.-q. The strategy research and method realization for the computer network flow control. *International Conference on Graphic and Image Processing*. 2013. URL: https://api.semanticscholar.org/CorpusID:62720079.

35. Gao F., Liu Q., Zhan H. Research of SIP DoS Defense Mechanism Based on Queue Theory. *International Conference on Computer Science, Environment, Ecoinformatics, and Education*. 2011. URL: https://api.semanticscholar.org/CorpusID:14887144.

36. Minkevius S., Sakalauskas L. On the law of iterated logarithm for extreme queue length in an open queueing network. *International Journal of Computer Mathematics: Computer Systems Theory*. 2021. Vol. 6. P. 220 - 235. URL: https://api.semanticscholar.org/CorpusID:237548367.

37. Stuckey N. Stochastic Estimation and Control of Queues within a Computer Network. 2012. URL: https://api.semanticscholar.org/CorpusID:59897689.

38. RouteNet-Erlang: A Graph Neural Network for Network Performance Evaluation / M. F. Galmes et al. *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*. 2022. P. 2018-2027. URL: https://api.semanticscholar.org/CorpusID:247159041.

39. The Analysis of Performance for Priority Distinction Double-queue and Double-server Communication Network / J. L. Xiong et al. *International Journal of Communication*. 2014. Vol. 3. URL: https://api.semanticscholar.org/CorpusID:62382654.

40. Andreji M. Analysis of the influence of queue type on packet delay in computer networks. 2015. URL: https://api.semanticscholar.org/CorpusID:195971842.

41. Queue Control Model in a Clustered Computer Network using M/M/m Approach / A. Ejem et al. *International Journal of Computer Trends and Technology*. 2016. Vol. 35. P. 12-20. URL: https://api.semanticscholar.org/CorpusID:38081651.

42. Розрахунок ефективності використання обчислювальних ресурсів самовідновлювальної комп'ютерної системи / N. Kuchuk та ін. *Системи управління, навігації та зв'язку. Збірник наукових праць*. 2021. Т. 3, № 65. С. 92–95. URL: 10.26906/sunz.2021.3.092.

43. Poslaiko N. I. Дослідження стаціонарного режиму в системі масового обслуговування типу gi /m /1 зі слабкою післядією. *Problems of applied mathematics and mathematic modeling*. 2022. URL: 10.15421/322119.

44. Порівняльна ефективність класифікаторів зображень під час розпізнавання зон інтересу при лапароскопічних втручаннях / М. Р. Баязітов та

ін. Medical Informatics and Engineering. 2020. № 2. С. 62–69. URL: 10.11603/mie.1996-1960.2020.2.11175.

45. Франчук Н. П. Стан та перспективи технологій машинного перекладу тексту. Theory and methods of e-learning. 2014. Т. 3. С. 319–325. URL: 10.55056/e-learn.v3i1.356.

46. Ярошенко О. Огляд інструментів ocr для завдання розпізнавання таблиць і графіків у документах. Information, Computing and Intelligent systems. 2022. № 3. URL: 10.20535/2708-4930.3.2022.265200.

47. Денисенко О. Дослідження та розробка системи розпізнавання тексту на зображенні за допомогою згорткової нейронної мережі. LES TENDANCES ACTUELLES DE LA MONDIALISATION DE LA SCIENCE MONDIALE - VOLUME 1. 2020. URL: 10.36074/03.04.2020.v1.30.

48. Польшакова О., Мальченко Є. Представлення ефективного методу розпізнавання тексту на зображенні, заснованого на критерії схожості структурних моделей символів. Адаптивні системи автоматичного управління. 2021. Т. 1, № 38. С. 50–56. URL: 10.20535/1560-8956.38.2021.233184.

49. Tibshirani R., Walther G., Hastie T. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*. 2001. Vol. 63, no. 2. P. 411--423.

50. Takeshita K., Shioyama J., Uchida S. Label or message: A large-scale experimental survey of texts and objects co-occurrence. *Proceedings of the International Conference on Pattern Recognition (ICPR)*. 2021. P. 6227--6234.