Міністерство освіти і науки України

Національний університет „Львівська політехніка”

Кафедра СКС



**Пояснювальна записка**

**до курсової роботи**

*з курсу:*

„Обробка сигналів і зображень”

*на тему:*

„Розробка процесора ШПФ”

Виконав:

ст. гр. КІ-409

Павлось М.Ю.

Прийняв:  
к.т.н., доцент

Ваврук Є.Я.

Львів – 2024

# Завдання

1. **Дослідити теоретичні основи Швидкого Перетворення Фур’є (ШПФ):**

* Порівняти Дискретне Перетворення Фур’є (ДПФ) та Швидке Перетворення Фур’є (FFT) з точки зору обчислювальної складності.
* Розглянути основні алгоритми FFT: Radix-2, Radix-4, прорідження по часу (time decimation), прорідження по частоті (frequency decimation).
* Обґрунтувати вибір алгоритму (Radix-4 або Radix-2) для розмірності N=1024N=1024N=1024.

1. **Розробити структуру та принципи функціонування процесора ШПФ, що відповідає наступним вимогам:**

* Розмірність ШПФ: N=1024N = 1024N=1024.
* Тип прорідження: часове (T).
* Використання віконної (вагової) функції Блекмана для зменшення спектрального витікання.
* Розрядність вхідних даних: 24 біти, розподілені як 12 біти цілої частини та 12 біти дробової (формат Q12.12).
* Цільовий час обробки блоку (0,4 мс).
* Вхідний інтерфейс: HOST (імітація прийому даних від хост-пристрою).
* Вихідний інтерфейс: USB (імітація передачі результатів).

1. **Реалізувати (або змоделювати) алгоритм ШПФ для заданого формату даних (Q12.12):**

* Забезпечити бітове дзеркальне переставлення індексів (bit-reversal).
* Продемонструвати розрахунок вагових коефіцієнтів (коренів з одиниці) для етапів метеликів (butterfly).
* Застосувати вікно Блекмана до вхідних даних перед обчисленням FFT.
* Здійснити обчислення FFT у фіксованій комі, враховуючи масштабування при множенні.
* Оцінити точність результатів та визначити, як впливає обмежена розрядність на похибку обчислень.

1. **Імітувати роботу інтерфейсів HOST та USB на рівні програмного коду:**

* Організувати читання вхідних даних (HOST) із зовнішнього джерела (наприклад, файл) або генерування тестового сигналу (синусоїда, шум тощо).
* Вивести результати (USB) до зовнішнього середовища (наприклад, у файл), з можливістю подальшого аналізу.

1. **Забезпечити можливість перевірки роботи процесора ШПФ на тестовому сигналі:**

* Згенерувати синусоїдальний сигнал, зашумлений чи без шуму, застосувати вікно Блекмана, виконати FFT та проаналізувати отриманий спектр.
* Визначити положення спектральних піків у залежності від частоти тестового сигналу.

1. **Оцінити продуктивність та ресурси, необхідні для реалізації в реальному пристрої:**

* Розрахувати обсяг пам’яті для зберігання вхідних даних, віконних коефіцієнтів та вагових коефіцієнтів метеликів.
* Оцінити кількість операцій (множень і додавань) для одного блоку довжиною 1024 та порівняти з цільовим часом обробки (0,4 мс).
* Пояснити, як даний проект можна перенести на платформу DSP або FPGA для досягнення необхідної швидкодії.

1. **Оформити пояснювальну записку з детальним описом:**

* Теоретичних відомостей про FFT та віконні функції.
* Структури та алгоритму роботи процесора ШПФ (у тому числі, архітектури або структурної схеми апаратної реалізації, якщо передбачено).
* Програмної реалізації (листинги коду, пояснення основних функцій).
* Результатів тестування на прикладі штучно згенерованих сигналів.
* Висновків щодо точності, продуктивності та перспектив розвитку.

# Анотація

У даній курсовій роботі представлено розробку процесора Швидкого Перетворення Фур’є (ШПФ) для обробки блоків даних розміром N = 1024. Застосовано часову схему прорідження (time decimation) та віконну функцію Блекмана для зменшення спектральних витікань. Вхідні дані мають розрядність 24 біти (12+12), що моделюється у фіксованій комі (Q12.12). Розглянуто архітектурні особливості проєкту, а також реалізовано інтерфейси приймання (HOST) та передачі (USB) даних. У пояснювальній записці наведено теоретичні основи FFT, описано структуру та алгоритм роботи процесора, а також проаналізовано час обробки і ресурси, необхідні для практичної реалізації на базі DSP або FPGA. Одержані результати підтверджують доцільність використання даного підходу для високопродуктивного спектрального аналізу.

**Зміст**

[Вступ 5](#_Toc182576533)

[1. Аналітичний огляд 6](#_Toc182576534)

[1.1. Основні характеристики зображення та їх обробки 6](#_Toc182576535)

[1.2 Аналіз алгоритмів для розпізнавання зображеннь 14](#_Toc182576536)

[1.3. Відомі програмні засоби 19](#_Toc182576537)

[2. Розробка алгоритмів 21](#_Toc182576538)

[2.1. Обробка зображення 21](#_Toc182576539)

[2.2. Розпізнавання символів в Tesseract 27](#_Toc182576540)

[2.3. Параметри зображень для розпізнавання 28](#_Toc182576541)

[3. Програмна реалізація 29](#_Toc182576542)

[3.1 Модуль обробки зображення 30](#_Toc182576543)

[3.2 Модуль розпізнавання тексту 34](#_Toc182576544)

[3.3 Модуль очищення тексту 35](#_Toc182576545)

[3.4 Модуль графічного інтерфейсу 36](#_Toc182576546)

[4. Визначення якості розпізнавання 39](#_Toc182576547)

[Висновки 40](#_Toc182576548)

[Список літератури. 41](#_Toc182576549)

[Додатки 43](#_Toc182576550)

[Додаток А 43](#_Toc182576551)

[Додаток B 50](#_Toc182576552)

# Вступ

Цифрова обробка сигналів (ЦОС) нині є невід’ємною частиною широкого спектра прикладних завдань: від аудіо- та відеообробки до телекомунікацій, радіолокації та вимірювальної техніки. Серед ключових інструментів ЦОС вирізняється Дискретне Перетворення Фур’є (ДПФ), яке дає можливість аналізувати й обробляти сигнали у частотній області. Втім, пряме обчислення ДПФ має високу обчислювальну складність , що робить його малопридатним для використання у реальному часі й особливо для великих розмірностей *N*.

Швидке Перетворення Фур’є (ШПФ, англ. FFT) суттєво зменшує кількість необхідних обчислювальних операцій (до порядку ), завдяки чому можна досягти високої продуктивності під час перетворення великих блоків даних. Це відкриває можливості застосовувати перетворення Фур’є у режимі реального часу — у випадках, коли затримка обробки сигналу повинна бути мінімальною.

Метою цієї курсової роботи є розробка та моделювання процесора ШПФ із конкретними параметрами:

* Розмірність перетворення .
* Віконна функція Блекмана для підготовки сигналу та зменшення спектральних витікань.
* Розрядність даних 24 біти (12 біт цілих + 12 біт дробових), що дозволяє зберігати сигнал у фіксованій комі Q12.12.
* Часове прорідження (T) як спосіб реалізації алгоритму FFT.
* Імітація вхідного (HOST) та вихідного (USB) інтерфейсів для прийому та передачі даних.

У роботі подається теоретичне підґрунтя ШПФ, розглянуто особливості застосування віконних функцій, а також описано методи програмної реалізації з урахуванням фіксованої коми. Передбачено можливість апаратної реалізації процесора на DSP або FPGA за необхідності. Основна увага приділена оптимізації алгоритму та аналізу часових характеристик, що підтверджують здатність обробляти блок сигналу довжиною 1024 за 0,4 мс, як передбачено умовами завдання.

# 1. Аналітичний огляд

## 1.1. Основні характеристики зображення та їх обробки

Зображення відіграють ключову роль у багатьох сучасних технологіях, особливо в тих, що пов'язані з обробкою візуальної інформації. Вивчення основних характеристик зображення є необхідною передумовою для розробки ефективних алгоритмів їх обробки, зокрема для розпізнавання тексту за допомогою технологій OCR (Optical Character Recognition). Характеристики зображень можуть суттєво впливати на точність результатів розпізнавання, і розуміння цих параметрів є критичним для ефективної роботи алгоритмів.

**Роздільна здатність**

Рекомендована роздільна здатність для OCR — мінімум 300 DPI, що дозволяє детальніше відображати символи, знижуючи ймовірність помилок розпізнавання [10]. Для менших шрифтів краще застосовувати 400 DPI. Низька роздільна здатність погіршує якість зображення, знижуючи точність OCR, тоді як надмірно висока роздільна здатність може уповільнити процес обробки зображень [18].

**Яскравість**

Налаштування яскравості в сканерах збалансовує світлі та темні відтінки у відсканованих зображеннях. Занадто високі або занадто низькі налаштування яскравості можуть зробити деякі дані нечіткими, знижуючи точність розпізнавання. Тому для всіх вимог до сканування рекомендується встановлювати яскравість за замовчуванням на рівні 50% [18].

**Конвертація зображення в градації сірого**

Перетворення кольорового зображення в градації сірого є важливим етапом у обробці зображень для подальшого аналізу, наприклад, при розпізнаванні тексту за допомогою OCR. Головною метою цього процесу є зменшення кількості інформації, зберігаючи лише яскравість (лінійну інформацію) для подальшої обробки.

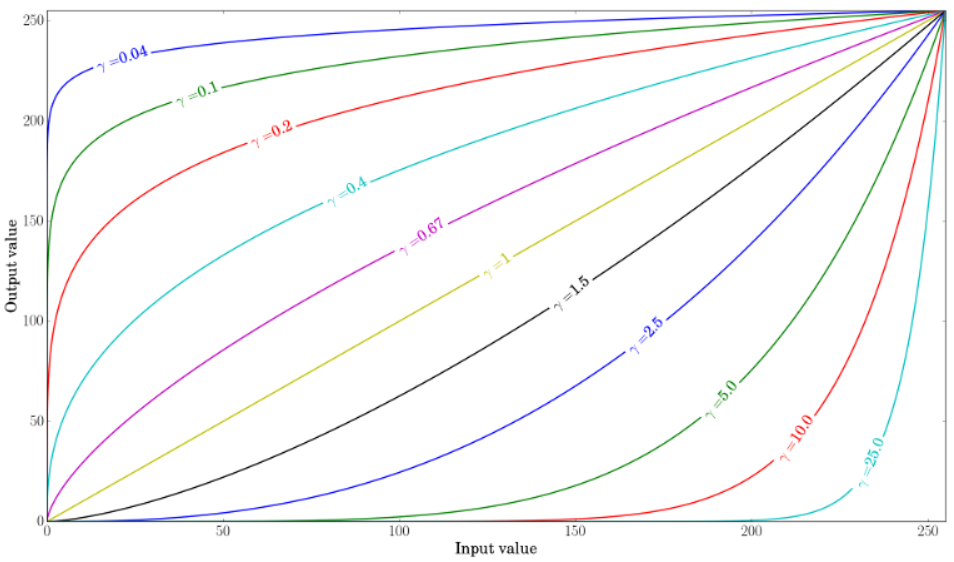
Оскільки людський зір не сприймає всі кольори однаково, внесок основних кольорів змінюється. Це залежить від довжини хвиль кольорів [25]. Часто для перетворення застосовуються вагові коефіцієнти, що визначають внесок кожного кольору в загальну яскравість. У випадку BGR це може виглядати так: [15][25]

Y←0.114⋅B +0.587⋅G+0.299⋅R

**Гамма-корекція**

Гамма-корекція допомагає регулювати яскравість та контраст, роблячи зображення більш зручним для OCR. Вона дозволяє краще зберігати деталі як у світлих, так і у темних областях, що покращує роздільну здатність та якість розпізнавання тексту. Регулюючи значення гами, можна ефективно контролювати яскравість і контрастність зображення, що призводить до більш точного відтворення кольорів [7].

Оскільки ця залежність нелінійна, ефект не буде однаковим для всіх пікселів і залежатиме від їхнього початкового значення.

*  
Рис.1. Графік для різних значень гами*

Коли γ < 1, вихідні темні області будуть яскравішими, а гістограма буде зсунута вправо, тоді як при γ > 1 все буде навпаки.[15]

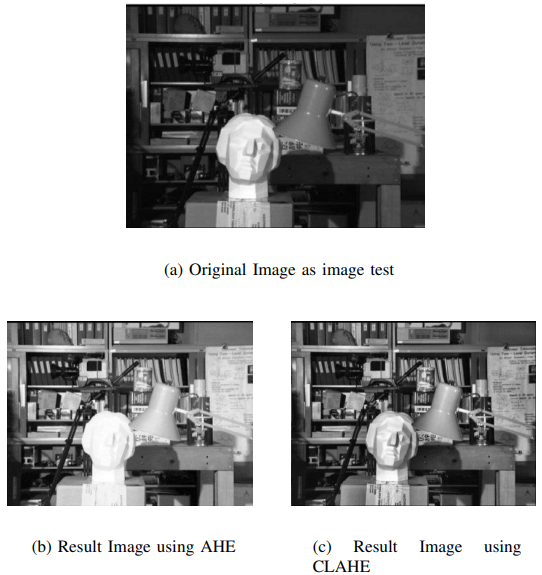
**Контраст**

Контраст між текстом і фоном має бути високим, щоб OCR легко відрізняв символи. Наприклад, чорний текст на білому фоні є оптимальним варіантом. Низький контраст або колірний шум може суттєво знизити точність розпізнавання [10].

**Метод CLAHE**

Адаптивне гістограмне вирівнювання (AHE) - це метод попередньої обробки зображень, який використовується для покращення контрастності зображень. Він обчислює кілька гістограм, кожна з яких відповідає окремій ділянці зображення, і використовує їх для перерозподілу значень яскравості зображення. Тому він підходить для покращення локального контрасту та покращення визначення країв у кожній області зображення. Однак AHE має тенденцію до надмірного посилення шуму у відносно однорідних ділянках зображення. Варіант адаптивного гістограмного вирівнювання, який називається адаптивне гістограмне вирівнювання з обмеженням контрасту (CLAHE), запобігає цьому ефекту, обмежуючи підсилення. [14]

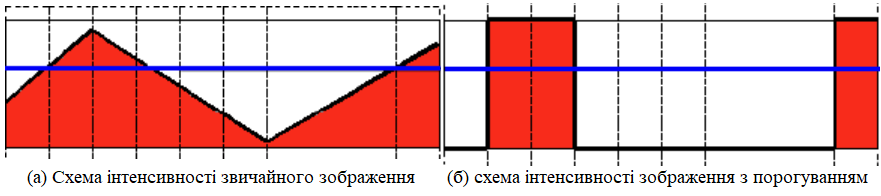
CLAHE - Адаптивне вирівнювання гістограми працює таким чином, що зображення розбивається на невеликі блоки, які називаються «плитками». Потім кожен з цих блоків вирівнюється гістограмою, як зазвичай. Таким чином, у невеликій області гістограма буде обмежуватися невеликою областю (якщо немає шуму). Якщо шум є, він буде посилений. Щоб уникнути цього, застосовується обмеження контрастності. Якщо будь-який бін гістограми перевищує задану межу контрастності, ці пікселі відсікаються і рівномірно розподіляються по інших бінах перед застосуванням вирівнювання гістограми. Після вирівнювання, для усунення артефактів на границях плиток, застосовується білінійна інтерполяція. [15]

*  
Рис.2. Зображення з методами AHE та CLAHE*

**Порогування та метод Оцу**

Найпростіший базовий метод пороговуння: До кожного пікселя застосовується однакове порогове значення. Якщо значення пікселя менше або дорівнює пороговому, воно встановлюється в 0, інакше - в максимальне значення.[15] Це можна виразити у такій формі:

де *src(x, y)* – значеннями вхідного зображення,  
а *thresh* – фіксований поріг

*  
Рис.3. Схема принципу роботи порогування*

При глобальному пороговому значенні ми використовували довільно вибране значення як поріг. На відміну від цього, метод Оцу уникає необхідності вибирати значення і визначає його автоматично.

Обчислювальне рівняння можна описати так:

,

де та - це ймовірності двох класів, поділені на поріг а та – дисперсія цих двох класів [18], значення якого знаходиться в діапазоні від 0 до 255 включно.

Ймовірність обчислюється з L гістограм:

Існує два способи знайти поріг. [11] Перший - мінімізувати внутрішньокласову дисперсію, визначену вище, другий - максимізувати міжкласову дисперсію, використовуючи вираз, наведений нижче:

, де – середнє значення класу *i*.

Це виражається в термінах ймовірностей класу ω та середнє значення класу μ. Середнє значення класу , та дорівнюють:

Наступні співвідношення можна легко перевірити:

Ймовірності класу і засоби класу можуть бути обчислені ітеративно. Ця ідея дає ефективний алгоритм.

Слід зазначити, що якщо поріг було обрано неправильно, то дисперсія певного класу може бути великою. Щоб отримати загальну дисперсію, потрібно підсумувати внутрішньокласові та міжкласові дисперсії: , де . Загальна дисперсія зображення () не залежить від порогу. [11]

**Шуми**

Шуми — це випадкові варіації інтенсивності пікселів, які можуть виникати в зображенні через умови зйомки, якість обладнання або процес цифрового перетворення. Шуми можуть спотворювати зображення і заважати точному аналізу.

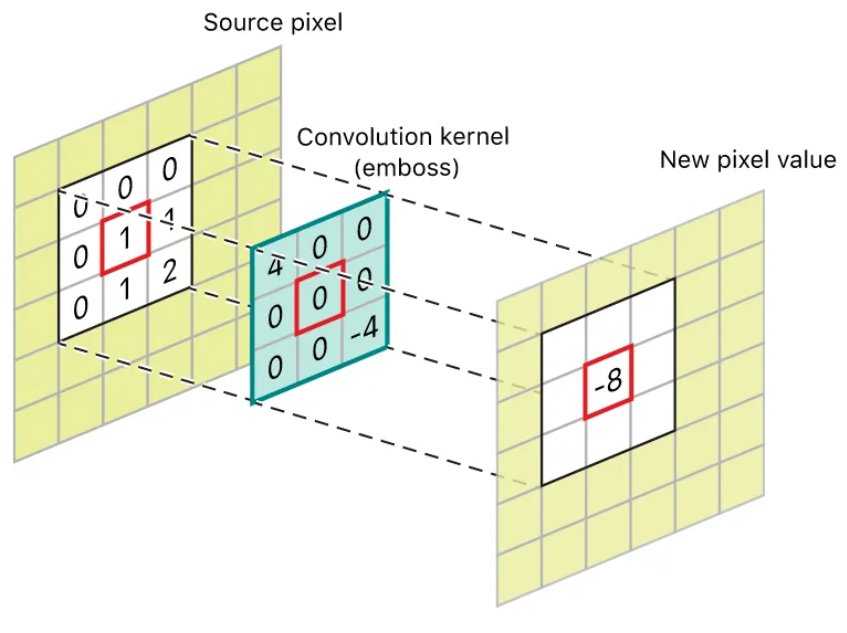
**Гауссове згладжування**

Гаусова фільтрація виконується шляхом згортання кожної точки у вхідному масиві за допомогою гаусового ядра , а потім підсумовування їх усіх для отримання вихідного масиву. Розмір ядра та його значення визначають ступінь і силу розмиття, застосованого до зображення [8]. Метод Гауса добре справляється зі зменшенням шуму, зберігаючи при цьому загальний зміст зображення. Він фактично видаляє високочастотний вміст (наприклад, шум, краї) із зображення. Таким чином, краї трохи розмиваються під час цієї операції (існують також методи розмивання, які не розмивають краї). [15]

Можна сказати, що Гауссове згладжування — це просто метод згладжування зображення за допомогою функції Гауса. [21] Формула Гауса у двох вимірах:

деσ визначає ступінь розмиття. Збільшення σ розширює область впливу фільтра, збільшуючи силу розмиття.

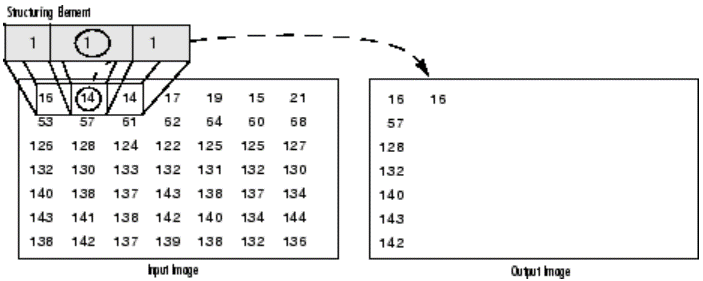
Значення функції Гауса створюють матрицю або ядро згортки, яке застосовується до кожного пікселя оригінального зображення. Зазвичай ядро невелике, оскільки збільшення його розміру вимагає більше обчислень для кожного пікселя. Процес згортки відбувається шляхом розміщення центру ядра на кожному пікселі та обчислення середньозваженого значення інтенсивності пікселів навколо нього. Ваги розподіляються за Гаусом: центральний піксель отримує найбільшу вагу, тоді як сусідні пікселі мають поступово менші значення. Такий підхід до зваженого усереднення забезпечує плавне накладання та ефективне розмивання зображення, мінімізуючи різкі зміни значень пікселів. [8][21]

*  
Рис.4. Застосування рельєфного ядра для згортки*

**Морфологічні операції**

Морфологія - це набір операцій обробки зображень, які обробляють зображення на основі попередньо визначених структурних елементів, відомих також як ядра. Значення кожного пікселя у вихідному зображенні базується на порівнянні відповідного пікселя у вхідному зображенні з його сусідами. Вибираючи розмір і форму ядра, можна побудувати морфологічну операцію, чутливу до конкретних форм вхідного зображення. [15]

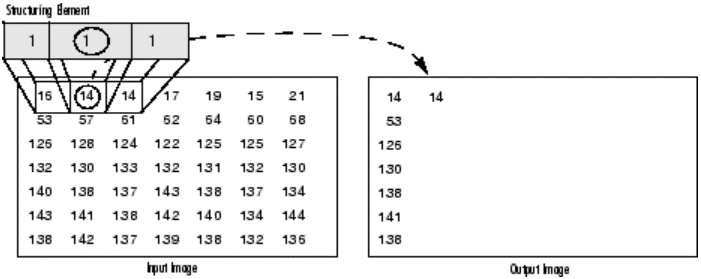
Дві найпростіші морфологічні операції - це дилатація та ерозія. Дилатація додає пікселі до меж об'єкта на зображенні, тоді як ерозія робить прямо протилежне. Кількість доданих або видалених пікселів, відповідно, залежить від розміру та форми структурного елемента, який використовується для обробки зображення.[14] Загалом, правила, що випливають з цих двох операцій, такі:

*  
Рис.5. Дилатація сірого зображення*

Дилатація множини A по множині B визначається як , де A - двійкове зображення, а B - структурний елемент.

Дилатацію також можна отримати за формулою , де позначає симетричну відносно B.

Ерозія: Для операції ерозії застосовується зворотна операція. [15] Значення вихідного пікселя - це мінімальне значення всіх пікселів, які підпадають під розмір і форму структурного елемента (рис.6).

*  
Рис.6. Ерозія сірого зображення*

Ерозія визначається як , де - трансляція *B* вектором z, тобто, . [9]

Дилатація та ерозія часто використовуються в комбінації для реалізації операцій обробки зображень.

**Формат файлу та стиснення**

Серед двох типів стиснення зображень - з втратами та без втрат - для кращого розпізнавання OCR рекомендується використовувати останній. Оскільки у форматі файлів із втратами частина даних відкидається, зменшуючи загальний обсяг даних, його не рекомендується використовувати для важливих документів. Формати файлів із втратами, такі як JPEG, мають невеликий розмір, що полегшує зберігання багатьох документів. Однак при стисненні без втрат файли зменшуються без втрати даних.

Інші типи документів зберігають відскановані зображення у форматі TIFF або PNG без стиснення. Вони забезпечують кращу подальшу обробку [18].

## 1.2 Аналіз алгоритмів для розпізнавання зображеннь

Існує кілька підходів до розпізнавання тексту на зображеннях, кожен з яких має свої переваги та недоліки. Більшість сучасних алгоритмів для OCR базуються на методах комп'ютерного зору та штучного інтелекту. Серед найпопулярніших підходів можна виділити методи на основі шаблонів, алгоритми сегментації та гібридні методи з використанням машинного навчання.

**Алгоритми на основі шаблонів**

Алгоритми на основі шаблонів (Template Matching) є одними з найстаріших методів OCR. Більшість алгоритмів розпізнавання символів за зразком потребують шаблонів відомих символів. Вони порівнюються зі сканованим зображенням для ідентифікації та інтерпретації символів на ньому. Основна перевага цього підходу полягає в його простоті. Інші методи потребують попереднього навчання на відомих наборах даних, щоб створити розуміння кожного символу і розвинути здатність розпізнавати кожен символ. Таке попереднє налаштування зазвичай виконується шляхом подачі алгоритмам відомого набору даних.[27] Однак недоліком є залежність від точності розпізнання шаблонів і їх обмежена здатність до адаптації до нових шрифтів або стилів тексту. Алгоритми на основі шаблонів добре працюють з друкованим текстом, але часто дають помилки при розпізнаванні рукописного тексту або зображень низької якості.

**Алгоритми сегментації**

Алгоритми сегментації передбачають поділ зображення на окремі символи або блоки тексту, що дозволяє локально аналізувати кожен символ окремо. Сегментація зображень ділить зображення на області з різними характеристиками і виокремлює області інтересу (ROI).[23] Цей підхід дає змогу досягти кращих результатів для тексту на складному фоні або при наявності шумів. Проте основним недоліком є складність налаштування алгоритму для різних типів зображень, оскільки він вимагає правильної сегментації.

**Методи глибокого навчання**

З останнім розвитком технологій машинного навчання, особливо з появою глибоких нейронних мереж, OCR-технології отримали новий імпульс у своєму розвитку. Одним із найуспішніших підходів є використання згорткових нейронних мереж (Convolutional Neural Networks, CNN), які дозволяють обробляти зображення з високою точністю.

CNN - це системи штучного інтелекту, засновані на багатошарових нейронних мережах, які можуть ідентифікувати, розпізнавати та класифікувати об'єкти, а також виявляти та сегментувати об'єкти на зображеннях, що робить їх надзвичайно корисними для задач OCR. На відміну від класичних алгоритмів на основі шаблонів, CNN не потребують явного визначення ознак або шаблонів для кожного символу. Вони здатні навчатися на великій кількості даних і самостійно визначати найважливіші ознаки для розпізнавання тексту. [24][1][22]

Алгоритми глибокого навчання також дозволяють вирішувати проблеми з рукописним текстом, який традиційно є більш складним для автоматичного розпізнавання. Поєднання згорткових нейронних мереж (CNN) з рекурентними нейронними мережами (RNN) дозволяє розпізнавати текст на зображеннях за допомогою CRNN. CNN витягує ознаки з вхідного зображення, тоді як RNN моделює послідовні залежності між символами в тексті. CRNN ефективно розпізнають текст у різних контекстах, включаючи розміри, стилі та орієнтацію шрифтів. Вони також можуть обробляти багато рядків тексту на зображенні і точно розпізнавати текст сцени. [1]

SVTR (Single Visual model for scene Text Recognition) - це вдосконалений движок розпізнавання тексту на зображеннях, який використовує комбінацію глибокого навчання та традиційних методів обробки зображень. Архітектура SVTR може розпізнавати сюжетний текст на фотографіях зі змінним освітленням, орієнтацією та нахилом. Крім того, вона має механізм уваги, який дозволяє мережі зосереджуватися на найважливіших візуальних компонентах, тим самим підвищуючи стійкість моделі до шуму та спотворень. [1]

Мережі з довгою короткочасною пам'яттю (long short-term memory, LSTM) - це тип рекурентних нейронних мереж (RNN), здатних вивчати довгострокові залежності. [6]

**Оптичне розпізнавання на основі трансформерів**

Оптичне розпізнавання символів (OCR) на основі трансформерів є сучасним підходом, що використовує архітектуру трансформерів для розпізнавання тексту в зображеннях. Цей метод поєднує можливості комп'ютерного зору та обробки природної мови, забезпечуючи ефективне перетворення зображень тексту в машинозчитуваний формат.

Трансформерні моделі OCR застосовують архітектуру трансформерів як для розуміння зображень, так і для генерації тексту на рівні словоформ. Вони складаються з двох основних компонентів:

Енкодер: перетворює вхідне зображення в послідовність векторів ознак.

Декодер: генерує текстову послідовність на основі векторів, отриманих від енкодера.

Однією з відомих моделей є TrOCR (Transformer-based Optical Character Recognition), яка використовує попередньо навчені моделі для обробки зображень та тексту. TrOCR спочатку змінює розмір вхідного зображення до 384×384 пікселів, розбиває його на послідовність із 16 патчів, які подаються на вхід трансформера. Стандартна архітектура трансформера з механізмом самопильності використовується як в енкодері, так і в декодері, де генеруються словоформи як розпізнаний текст із вхідного зображення. [12]

**Аналіз зв'язаних компонент**

**Connected component analysis** (**CCA**) - це метод обробки зображень, який використовується для групування пікселів на зображенні, що належать до одного об'єкта або сутності. Основна ідея полягає в тому, щоб визначити та позначити всі зв'язані компоненти на зображенні, де зв'язаний компонент - це набір пікселів, з'єднаних між собою за певним критерієм.

Найпоширенішим критерієм, що використовується для визначення зв'язних компонентів, є 4-зв'язність або 8-зв'язність. При **4-зв'язності** два пікселі вважаються з'єднаними, якщо вони мають спільний край, тоді як при **8-зв'язності** два пікселі вважаються з'єднаними, якщо вони мають спільний край або кут. [4]

### Переваги та недоліки різних підходів

Кожен з наведених алгоритмів має свої переваги та недоліки, що робить їх придатними для різних типів задач.

Алгоритми на основі шаблонів:

* **Переваги**: простота реалізації, висока точність для друкованого тексту.
* **Недоліки**: погана адаптація до нових шрифтів, низька точність при обробці рукописного тексту та зображень з шумами.

Алгоритми сегментації:

* **Переваги**: можливість працювати з різними типами тексту, підвищена точність на складному фоні.
* **Недоліки**: складність налаштування, необхідність додаткової обробки зображення.

Глибоке навчання:

* **Переваги**: висока точність, адаптація до різних шрифтів, можливість обробки рукописного тексту.
* **Недоліки**: велика потреба у обчислювальних ресурсах, залежність від якості навчальних даних.

Трансформери:

* **Переваги**: здатність враховувати контекст, висока точність для складних документів, можливість обробки структурованих документів.
* **Недоліки**: складність реалізації, потреба в значних обчислювальних потужностях.

**Аналіз зв'язаних компонент**

* **Переваги**: Простота і ефективність для виявлення ізольованих об'єктів на бінарних зображеннях, здатність сегментувати зображення на окремі області, що корисно для розпізнавання тексту та об'єктів, ефективне для малих і середніх за розміром зображень.
* **Недоліки**: Проблеми з обробкою зашумлених зображень, де компоненти можуть бути погано видимі; чутливість до порогових значень і якості зображення; зниження продуктивності на великих або складних зображеннях.

### Власний підхід

Мій підхід до розробки алгоритму розпізнавання тексту базується на поєднанні класичних та сучасних методів. Я планую використовувати попередню обробку зображень за допомогою OpenCV для підвищення якості зображень перед передачею їх до нейронних мереж. Це дозволить видалити шуми, підвищити контрастність і вирівняти геометричні спотворення. Для самого розпізнавання тексту я планую використовувати Tesseract, який використовує зараз архітектуру, що базується на LSTM.[24][6]

**Переваги Tesseract OCR:**

* **Переваги**:   
  Відкритий код і безкоштовність.  
  Підтримка декількох мов: підтрика понад 100 мов.  
  Висока точність друкованого тексту.  
  Налаштовуваність і розширюваність.  
  Підтримка багатьох плтформ.  
  Структурований вивід даних.
* **Недоліки**:  
  Обмежене розпізнавання рукописного тексту.  
  Проблеми зі складними макетами.  
  Залежність від якості.  
  Потребує попередньої обробки зображень. [4]

## 1.3. Відомі програмні засоби

**Google Cloud Vision OCR**

Google Cloud Vision OCR — це один із сервісів у рамках Google Cloud, який забезпечує потужне розпізнавання тексту з зображень за допомогою технологій штучного інтелекту та машинного навчання. Використовуючи потужні алгоритми комп'ютерного зору, цей сервіс може розпізнавати текст з фотографій, документів, сканів або будь-яких інших типів зображень і перетворювати його в цифровий формат. [3]

Плюси: висока точність розпізнавання, підтримка багатьох мов, робота з різними типами документів, інтеграція з іншими сервісами Google, надання розробникам API, широка функціональність.

Мінуси: ціна, необхідність інтернет з’єднання, швидкість залежить від інтернет-з’єднання.

**Capture2Text**

Безкоштовний інструмент для Windows, що дозволяє швидко розпізнавати текст зі знімків екрану. Підтримує велику кількість мов. Вона дуже проста у використанні і не потребує складної установки. [2]

Плюси: безкоштовність, простота використання, відносна швидкість, широкий діапазон мов.

Мінуси: обмежена точність, відсутність додаткових функцій, підтримка лише Windows.

**Manga OCR**

Це інструмент для оптичного розпізнавання символів, спеціально розроблений для роботи з манґою, враховуючи її особливості, такі як вертикальний та горизонтальний текст, фуріґана, різноманітні шрифти та стилі. [13]

Плюси: безкоштовність, відкритий код.

Мінуси: досить низька точність, підтримує лише одну мову, не універсальний.

**Apple Vision Framework**

OCR від Apple (Vision Framework) — це інструмент для розпізнавання тексту, вбудований у фреймворк Vision для операційних систем iOS, iPadOS та macOS. Він дозволяє розпізнавати текст із зображень або відеопотоків, що робить його потужним інструментом для розробників, які працюють в екосистемі Apple. Ця функція добре інтегрується в додатки та підтримує високу точність розпізнавання тексту з різноманітних джерел. [26]

Плюси: інтеграція з екосистемою Apple, розпізнавання в реальному часі, підтримка багатьох мов, автоматичне виявлення текстових блоків, розпізнавання рукописного тексту, локальна обробка.

Мінуси: обмежено під продукцію Apple, відсутність хмарної обробки.

**Tesseract OCR**

Відкритий код OCR-движка, спочатку розробленого HP, а нині підтримуваного Google. Tesseract є одним з найпопулярніших інструментів для текстового розпізнавання і підтримує безліч мов.

**Плюси**: Безкоштовний, підтримує безліч мов, висока точність.

**Мінуси**: може потребувати попередньої обробки, поганий варіант для рукописного тексту.

**ShareX OCR**

ShareX — це популярний безкоштовний інструмент для створення скріншотів і запису екрану, який має вбудовану функцію OCR для розпізнавання тексту. OCR-функціональність у ShareX дозволяє виділяти текст з зображень або скріншотів, після чого його можна редагувати чи копіювати.

Плюси: швидкість, безкоштовність, додаткові функції окрім OCR, користувацькі налаштування.

Мінуси: залежність від якості зображення, відсутність розширених функцій самого OCR.

# 2. Розробка алгоритмів

Цей розділ присвячений детальному розбору алгоритмів обробки зображень та розпізнавання тексту за допомогою OCR. У ньому розглянуто два основних етапи, що складають процес: перший — це попередня обробка зображення для підвищення якості тексту, а другий — безпосереднє розпізнавання символів за допомогою бібліотеки pytesseract, яка використовує популярний OCR-движок Tesseract.

## 2.1. Обробка зображення

**Конвертація зображення в градації сірого**

Конвертація зображення в градації сірого є першим етапом підготовки зображення для подальшої обробки. Основна ідея полягає в зменшенні кількості кольорів зображення до рівнів яскравості. Кожен піксель кольорового зображення має три канали (червоний, зелений, синій), а при перетворенні в сіре зображення кожен піксель отримує єдине значення інтенсивності, що відображає яскравість. [16]

  
Рис.7. Схема алгоритму грейскейлу

Алгоритм:

1. Вхідне кольорове зображення.

2. Обчислення інтенсивності для кожного пікселя (в цьому випадку для BGR) за формулою Y←0.114⋅B +0.587⋅G+0.299⋅R

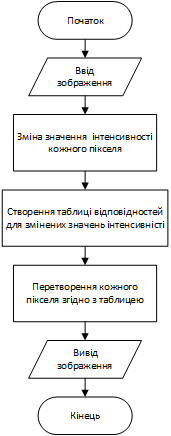
3. Формування нового сірого зображення.

4. Виведення сірого зображення.

OpenCV оптимізує цей процес шляхом векторизації обчислень для всіх пікселів одночасно. [15]

**Корекція гамми**

Наступним використовуватиму гамма-корекцію для налаштування яскравості зображення, що впливає на контраст і деталізацію.

*  
Рис. 8. Схема алгоритму гамма-корекції*

Алгоритм:

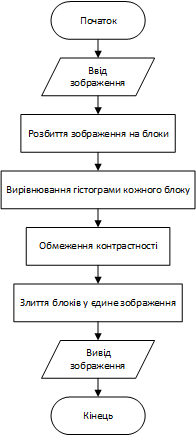
1. Для кожного пікселя значення інтенсивності змінюється за формулою:  
​, де γ – коефіцієнт гамма.

2. Створюється таблиця відповідностей (look-up table, LUT) для значень інтенсивності від 0 до 255 із застосуванням функції гамма.

3. Кожен піксель перетворюється згідно з таблицею.

**Метод CLAHE**

Контрастне адаптивне вирівнювання гістограми (CLAHE) призначене для покращення локального контрасту на зображенні. CLAHE розділяє зображення на невеликі блоки (tiles) і вирівнює гістограми для кожного блоку окремо.

 *Рис.9. Схема алгоритму CLAHE*

Алгоритм:

1. Зображення розбивається на невеликі області (tiles).

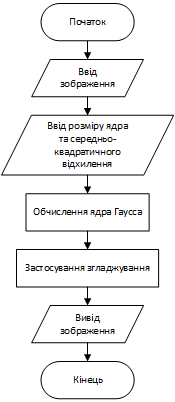
2. Для кожної області проводиться локальне вирівнювання гістограми.

3. Контрастність обмежується для запобігання надмірному посиленню шуму.

4. Вирівняні області зливаються назад в одне зображення.

**Гауссове згладжування**

Гауссове згладжування застосовується для зменшення шуму та підготовки зображення до порогування або розпізнавання символів. Цей фільтр розмиває зображення, зменшуючи дрібні деталі та шум, не сильно змінюючи контури об'єктів.

*  
Рис.10. Схема алгоритму Гауссового згладжування*

Алгоритм:

1. Створюється ядро Гаусса (двовимірне).

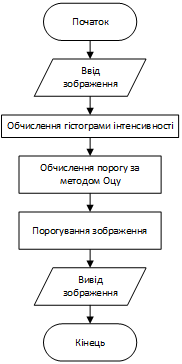
2. Зображення згладжується шляхом згортки з цим ядром, де кожен піксель замінюється зваженим середнім його сусідів згідно з ядром.

3. Значення пікселів навколо центру ядра мають більший вплив.

Потрібно вказати ширину та висоту ядра, які мають бути додатними та непарними. Також слід вказати середньоквадратичне відхилення в напрямках X та Y, sigmaX та sigmaY відповідно. Якщо вказано лише sigmaX, то sigmaY приймається рівним sigmaX. Якщо обидві величини дорівнюють нулю, вони обчислюються за розміром ядра.

**Порогування та метод Оцу**

Метод Оцу мінімізує внутрішньокласову дисперсію яскравості для двох класів — об'єктів і фону, автоматично знаходячи оптимальний поріг.

*  
Рис.11. Схема алгоритму Оцу*

Алгоритм для варіанту максимізації міжкласової дисперсії можна представити наступним чином:[18]

1. Обчислення гістограму та ймовірність кожного рівня інтенсивності.

2. Установка початкових та .

3. Проходження усі можливі порогові значення t = 1, … до максимального.

1. Оновлення та .

2. Обчислення .

4. Бажаний поріг відповідає максимуму.

Обчислення порогу за методом Оцу відбуваються за формулами: [11]

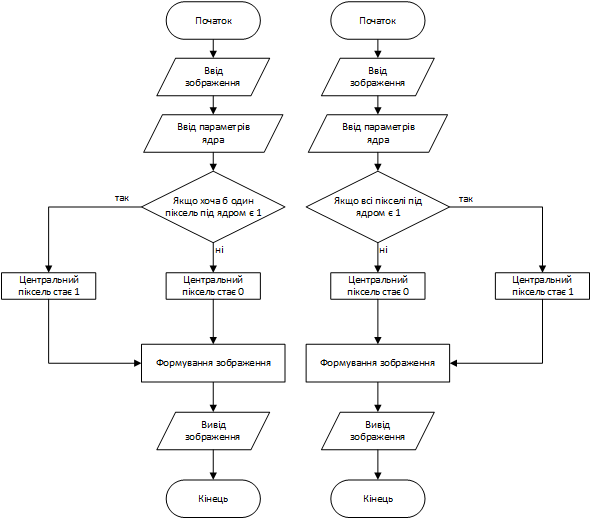
Ймовірність обчислюється з L гістограм:

Середнє значення класу , та дорівнюють:

Щоб отримати загальну дисперсію, потрібно підсумувати внутрішньокласові та міжкласові дисперсії: , де . [17]

**Морфологічні операції**

Я буду вкористовувати закриття (спочатку розширення, потім звуження) — заповнює отвори всередині й на межах.

  
Рис.12 Схема алгоритму дилатації та ерозії

На рис.12 зліва знаходиться схема алгоритму дилатації, а зліва – схема алгоритму ерозії.

Алгоритм морфологічного замикання:

Дилатація: Спочатку виконується розширення об'єктів, заповнюючи невеликі порожнини та об'єднуючи розірвані частини об'єктів. Якщо хоча б один піксель під ядром є 1, центральний піксель стає 1.

Ерозія: Після дилатації застосовується ерозія, яка згладжує розширені об'єкти, видаляючи зайві пікселі, але при цьому зберігаючи структуру, що була об'єднана в процесі дилатації. Якщо всі пікселі під ядром є 1, центральний піксель залишається 1. Якщо хоча б один — 0, центральний піксель стає 0.

## 2.2. Розпізнавання символів в Tesseract

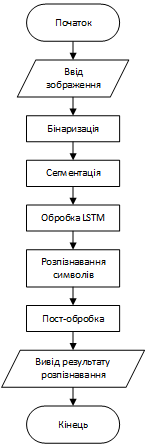
  
Рис.13. Схема алгоритму розпізнавання тексту Tesseract OCR

Схема знизу (рис.11) демонструє архітектуру процесу розпізнавання тексту в OCR (оптичному розпізнаванні символів) системі Tesseract, яка складається з кількох основних етапів. [19]

**Обробка LSTM:** Головний блок системи, який отримує оброблене зображення, виконує процес розпізнавання символів, використовуючи набори навчених моделей (trained dataset), де знаходяться дані для різних мов і шрифтів. Ці моделі допомагають OCR-движку розпізнавати текст із високою точністю.

**Постобробка (Post-Processor):** Після розпізнавання проводиться постобробка тексту для виправлення можливих помилок, покращення точності, форматування, і корекції з використанням словників та мовних моделей.

## 2.3. Параметри зображень для розпізнавання

Формат зображень jpg. Розміри зображень 4000 на 3000. 96 DPI. Бітова глибина 24. Розмірнвсть зображень 1-4 Mb.

Більшість зображень будуть зображеннями з книг. Кількість символів на зображеннях різна. Є зображення як і до 100 символів, так і більше ніж 500. Зображення не найкращої якості, багато без фокусу, також на багатьох є затемнення та засвітлення для перевірки як програма буде поратись з такими зображеннями. Звісно, де якість була краща виявся кращим і результат.

# 3. Програмна реалізація

Я обрав мову Python тому, що вона ідеально підходить для реалізації задач з обробки зображень та розпізнавання тексту. Python має зручні бібліотеки для OCR, обробки зображень і створення графічного інтерфейсу, такі як OpenCV, Pytesseract та Tkinter. Python також є кросплатформним, дозволяє швидко тестувати код та підтримує великий вибір модулів для обробки тексту і графіки.

Зокрема:

OpenCV забезпечує ефективну роботу з зображеннями, пропонуючи широкий вибір інструментів для попередньої обробки.

Pytesseract інтегрується з Tesseract OCR, забезпечуючи точне розпізнавання тексту на основі мовних налаштувань.

Tkinter дозволяє швидко створити графічний інтерфейс користувача для вибору зображення, налаштувань і перегляду результатів.

Інші бібліотеки (такі як re для обробки тексту, difflib для порівняння тексту, PIL для роботи із зображеннями) доповнюють функціонал і дозволяють реалізувати специфічні вимоги до очистки тексту, форматування та порівняння.

Лістинг програми наведений в додатку.

Програма складається з таких основних модулів:

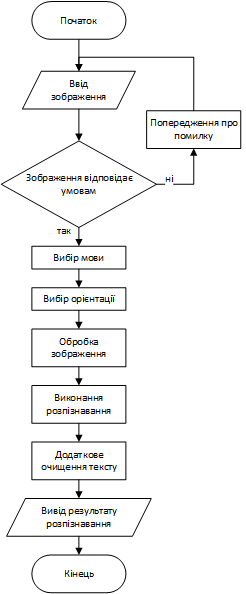
модуль обробки зображення;

модуль текстового розпізнавання;

модуль очищення тексту;

модуль перевірки точності;

графічний інтерфейс користувача;



## 3.1 Модуль обробки зображення

Модуль, що виконує головну функцію програми, тут виконується оброблення зображення для того, щоб підготувати його до розпізнавання.

Обробка зображення відбувається в функції ocr\_from\_image, в якій також використовується функція adjust\_gamma для гамма-корекції та remove\_shadows для видалення тіней з зображення.

def adjust\_gamma(image, gamma=1.0):  
 *"""Функція корекції гамми."""* inv\_gamma = 1.0 / gamma  
 table = np.array([((i / 255.0) \*\* inv\_gamma) \* 255 for i in np.arange(0, 256)]).astype("uint8")  
 return cv2.LUT(image, table)  
  
def remove\_shadows(image):  
 *"""Функція видалення тіней з зображення."""* rgb\_planes = cv2.split(image) *# Розділити канали RGB* result\_planes = []  
 for plane in rgb\_planes:  
 *# Використання морфологічного закриття для оцінки фону* dilated\_img = cv2.dilate(plane, np.ones((7, 7), np.uint8))  
 bg\_img = cv2.medianBlur(dilated\_img, 21)  
 *# Віднімання фону від оригінального зображення* diff\_img = 255 - cv2.absdiff(plane, bg\_img)  
 result\_planes.append(diff\_img)  
  
 result = cv2.merge(result\_planes) *# Об'єднати оброблені канали* return result

def ocr\_from\_image(image\_path, language='ukr'):  
 *"""Основна функція обробки та розпізнавання тексту."""* image = cv2.imread(image\_path) *# Завантаження зображення* gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY) *# Перетворення в градації сірого* gamma\_corrected = adjust\_gamma(gray, gamma=0.6) *# Корекція гамми для зменшення засвітлення* clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8, 8)) *# Застосування CLAHE для покращення контрасту* enhanced\_image = clahe.apply(gamma\_corrected)  
 blurred = cv2.GaussianBlur(enhanced\_image, (3, 3), 0) *# Розмивання зображення (щоб зменшити шум)* \_, binary = cv2.threshold(blurred, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU) *# Бінаризація зображення для чіткості тексту* kernel = np.ones((1, 1), np.uint8) *# Застосування морфологічних операцій для покращення текстових об'єктів* dilated = cv2.dilate(binary, kernel, iterations=1)  
 eroded = cv2.erode(dilated, kernel, iterations=1)

Основні кроки:

1. Завантаження зображення: Читає зображення з файлу за допомогою cv2.imread().

2. Видалення тіней: видалення тіні з зображення, використовуючи морфологічні операції для оцінки фону та його віднімання від оригінального зображення.

3. Перетворення в градації сірого: Конвертує зображення в чорно-біле (градації сірого) за допомогою cv2.cvtColor().

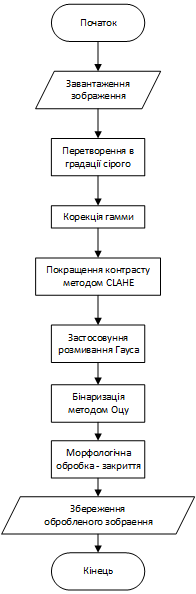
4. Корекція гамми: Використовує функцію adjust\_gamma для корекції яскравості, щоб зменшити пересвітлення на зображенні.

5. Покращення контрасту: Застосовує метод CLAHE (Contrast Limited Adaptive Histogram Equalization) для покращення контрасту на основі локальних областей зображення.

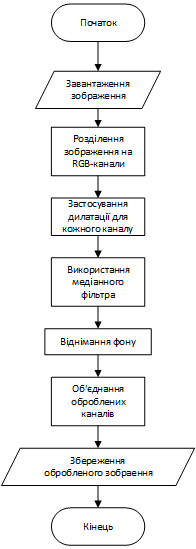
6. Розмивання зображення: Застосовує розмивання Гауса (cv2.GaussianBlur()), щоб зменшити шум перед подальшою обробкою.

7. Бінаризація: Перетворює зображення в бінарне (чорне та біле) за допомогою методу Оцу (cv2.threshold()), щоб зробити текст чіткішим.

8. Морфологічна обробка: Використовує морфологічні операції (дилатація та ерозія) для покращення структури тексту.

  
Рис.14. Схема алгоритму обробки зображення

Алгоритм функції видалення тіней виглядає так:

  
Рис.15. Схема алгоритму видалення тіней

**Розділення каналів зображення:**

rgb\_planes = cv2.split(image)

Зображення складається з трьох колірних каналів: червоний (R), зелений (G) та синій (B). Функція cv2.split розділяє зображення на ці три окремі канали.

**Обробка кожного каналу:** У циклі для кожного каналу (plane) виконується наступна обробка:

dilated\_img = cv2.dilate(plane, np.ones((7, 7), np.uint8))

Морфологічне розширення (dilation): Ця операція збільшує світлі області зображення (наприклад, тіні робляться більш рівними). За допомогою ядра розміру 7x7 функція cv2.dilate розширює світлі області, щоб оцінити фон.

bg\_img = cv2.medianBlur(dilated\_img, 21)

Медіанний фільтр: Використовується для згладжування зображення та отримання більш рівного фону. Це допомагає "змити" шуми та дрібні деталі.

diff\_img = 255 - cv2.absdiff(plane, bg\_img)

Віднімання фону: Після оцінки фону відбувається його віднімання від оригінального каналу. Це робиться за допомогою функції cv2.absdiff, яка обчислює абсолютну різницю між оригінальним каналом та фоном. Потім ця різниця інвертується, щоб вирівняти освітлення і зробити тіні менш помітними.

**З'єднання каналів назад:**

result = cv2.merge(result\_planes)

Після обробки кожного каналу окремо, оброблені канали знову об'єднуються у повнокольорове зображення за допомогою функції cv2.merge.

## 3.2 Модуль розпізнавання тексту

Розпізнавання тексту також відбувається в функції ocr\_from\_image одразу після обробки зображення.

if is\_vertical and language == 'jpn':  
 custom\_config = f'--oem 3 --psm 5 -l jpn\_vert -c preserve\_interword\_spaces=1'  
else:  
 custom\_config = f'--oem 3 --psm 6 -l {language}'  
text = pytesseract.image\_to\_string(eroded, config=custom\_config)  
  
return text

Залежно від мови та напрямку тексту, модуль може використовувати різні конфігурації для підвищення точності розпізнавання.

*--oem 3* – Це параметр "OCR Engine Mode" (OEM), що задає, який алгоритм Tesseract буде використовувати:

*3* – Використовується лише LSTM (найчастіше це рекомендується, оскільки LSTM є більш точним). У цьому випадку --oem 3 вказує Tesseract використовувати тільки LSTM.

*--psm 6* або *--psm 5* – Це параметр "Page Segmentation Mode" (PSM), що визначає спосіб розділення сторінки:

*6* – "Assume a single uniform block of text," або "Єдиний блок тексту". Це означає, що Tesseract буде шукати текст як єдиний блок, ігноруючи наявність декількох колонок або розділів. Підходить для обробки простих текстів, де немає складного форматування.

*5* – "Assume that the text is aligned in a single column, but might be rotated vertically". Припускається, що текст має одну колонку, але може бути вертикальним.

*-l {language}* – Це параметр для визначення мови, де *{language}* – змінна, що містить код мови. Наприклад, якщо *language = "ukr"*, то Tesseract буде використовувати український мовний пакет для розпізнавання.

*-l jpn\_vert* – мова розпізнавання встановлюється на японську для вертикального тексту. Для вертикального японського тексту використовується спеціальний мовний пакет *jpn\_vert*.

*-c preserve\_interword\_spaces=1* – цей параметр вказує Tesseract зберігати пробіли між словами при розпізнаванні, що може бути важливим для деяких мов, де пробіли мають значення для структури тексту.

## 3.3 Модуль очищення тексту

Код містить три функції для очищення тексту, кожна з яких спрямована на видалення зайвих символів, виправлення пробілів та об'єднання фрагментованих слів для різних мов залежно від особливостей цієї мови.

def clean\_text\_japanese(text):  
 *"""Cleans up extra symbols, spaces, and joins fragmented words in Japanese text,  
 while retaining 'ー' only near Katakana."""* cleaned\_text = re.sub(r'[^a-zA-Z0-9\u3040-\u309f\u30a0-\u30ff\u4e00-\u9fff\s\.,!?\'"()-]', '', text) *# Remove unwanted symbols, keeping Japanese characters, Latin letters, digits, and some punctuation.* cleaned\_text = re.sub(r'\s\*([、。！？.,!?])\s\*', r'\1', cleaned\_text) *# Remove spaces around punctuation* cleaned\_text = re.sub(r'(?<=\w)\s+(?=\w)', '', cleaned\_text) *# Join fragmented words by removing spaces between Japanese characters* cleaned\_text = re.sub(r'(?<![\u30a0-\u30ff])ー(?![\u30a0-\u30ff])', '', cleaned\_text) *# Remove "ー" only when it’s not adjacent to Katakana* cleaned\_text = re.sub(r'([ー]{2,})', 'ー', cleaned\_text)  
 cleaned\_text = re.sub(r'\s+', ' ', cleaned\_text).strip() *# Final trim of excess whitespace* return cleaned\_text  
  
def clean\_text\_english(text):  
 *"""Очищує зайві символи та об’єднує розірвані слова для англійського тексту."""* cleaned\_text = re.sub(r'[^a-zA-Z0-9\s\.,!?\'"-]', '', text) *# Залишає літери, цифри та базові знаки пунктуації* cleaned\_text = re.sub(r'(?<=\w)\s+(?=\w)', ' ', cleaned\_text) *# Об’єднує слова з зайвими пробілами між ними* cleaned\_text = re.sub(r'\s\*([.,!?])\s\*', r'\1', cleaned\_text) *# Видаляє зайві пробіли перед і після знаків пунктуації* cleaned\_text = re.sub(r'\s+', ' ', cleaned\_text).strip() *# Видаляє зайві пробіли* return cleaned\_text  
  
def clean\_text\_ukrainian(text):  
 *"""Очищує зайві символи та об’єднує розірвані слова для українського тексту."""* cleaned\_text = re.sub(r'[^a-zA-Zа-яА-ЯіІїЇєЄ0-9\s\.,!?\'"-]', '', text)  
 cleaned\_text = re.sub(r'(?<=\w)\s+(?=\w)', ' ', cleaned\_text)  
 cleaned\_text = re.sub(r'\s\*([.,!?])\s\*', r'\1', cleaned\_text)  
 cleaned\_text = re.sub(r'\s+', ' ', cleaned\_text).strip()  
 return cleaned\_text

*clean\_text\_japanese*: Очищає японський текст, зберігаючи лише японські символи, латинські літери, цифри та основні знаки пунктуації. Видаляє зайві пробіли, фрагментовані слова та невірно розміщені символи "ー" (довга риска).

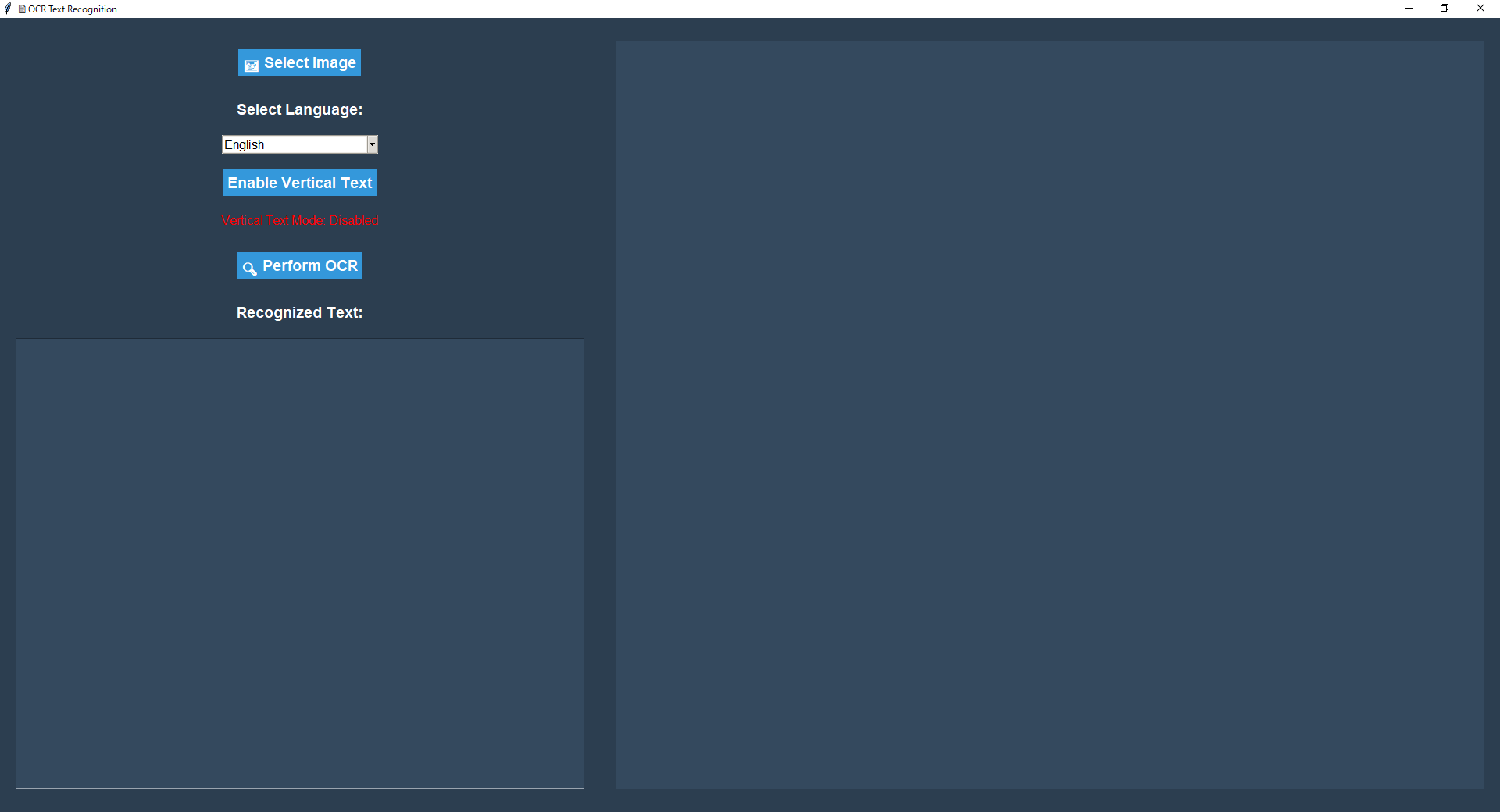
*clean\_text\_english*: Очищає англійський текст, зберігаючи латинські літери, цифри та стандартні знаки пунктуації. Об'єднує слова, видаляючи зайві пробіли та пробіли навколо пунктуації.

*clean\_text\_ukrainian*: Очищає український текст аналогічно англійському, зберігаючи кириличні літери, цифри та базову пунктуацію, видаляючи зайві пробіли та непотрібні символи.

## 3.4 Модуль графічного інтерфейсу

Код створює графічний інтерфейс (GUI) за допомогою бібліотеки Tkinter і її модуля ttk для оформлення елементів. Ось опис елементів інтерфейсу та їхнього розташування:

root = tk.Tk()  
root.title("📄 OCR Text Recognition")  
root.geometry(f"{root.winfo\_screenwidth()}x{root.winfo\_screenheight()}")  
root.state('zoomed')  
root.configure(bg='#2C3E50')  
  
image\_path = tk.StringVar()  
  
languages = {  
 'English': 'eng',  
 'Ukrainian': 'ukr',  
 'Japanese': 'jpn',  
}  
  
left\_frame = ttk.Frame(root, padding=20, style='TFrame')  
left\_frame.place(relx=0, rely=0, relwidth=0.4, relheight=1)  
  
right\_frame = ttk.Frame(root, padding=20, style='TFrame')  
right\_frame.place(relx=0.4, rely=0, relwidth=0.6, relheight=1)  
  
select\_button = ttk.Button(  
 left\_frame, text="🖼 Select Image", command=select\_image, style='Accent.TButton'  
)  
select\_button.pack(pady=20)  
  
language\_label = ttk.Label(left\_frame, text="Select Language:", font=('Helvetica', 14, 'bold'), background='#2C3E50', foreground='white')  
language\_label.pack(pady=10)  
  
language\_combobox = ttk.Combobox(left\_frame, values=list(languages.keys()), font=('Helvetica', 12))  
language\_combobox.set("English")  
language\_combobox.pack(pady=10)  
  
*# Add a checkbox to toggle vertical text recognition*vertical\_text\_var = tk.BooleanVar() *# Variable to store the state of the checkbox*vertical\_checkbox = ttk.Checkbutton(  
 left\_frame, text="Enable Vertical Text", variable=vertical\_text\_var, command=toggle\_vertical\_text,  
 style='Accent.TButton'  
)  
vertical\_checkbox.pack(pady=10)  
  
*# Label to display vertical text mode status*vertical\_status\_label = ttk.Label(left\_frame, text="Vertical Text Mode: Disabled", font=('Helvetica', 12), background='#2C3E50', foreground='red')  
vertical\_status\_label.pack(pady=10)  
  
ocr\_button = ttk.Button(  
 left\_frame, text="🔍 Perform OCR", command=perform\_ocr, style='Accent.TButton'  
)  
ocr\_button.pack(pady=20)  
  
*# Додаємо поле для виводу знайденого тексту*recognized\_text\_label = ttk.Label(left\_frame, text="Recognized Text:", font=('Helvetica', 14, 'bold'), background='#2C3E50', foreground='white')  
recognized\_text\_label.pack(pady=10)  
  
recognized\_text\_box = tk.Text(left\_frame, wrap='word', font=('Helvetica', 12), height=15, width=40, background='#34495E', foreground='white')  
recognized\_text\_box.pack(pady=10, fill='both', expand=True)  
  
image\_label = ttk.Label(right\_frame, background='#34495E', foreground='#2C3E50', font=('Helvetica', 10), anchor='center')  
image\_label.pack(pady=10, fill='both', expand=True)  
  
style = ttk.Style()  
style.theme\_use('clam')  
style.configure('TFrame', background='#2C3E50')  
style.configure('TLabel', background='#2C3E50', foreground='white', font=('Helvetica', 12))  
style.configure('Accent.TButton', background='#3498DB', foreground='white', font=('Helvetica', 14, 'bold'),  
 borderwidth=0)  
style.map('Accent.TButton', background=[('active', '#2980B9')])  
  
root.mainloop()

  
Рис.15. Графічний інтерфейс програми

Загальна структура

Інтерфейс складається з двох основних областей (фреймів):

Лівий фрейм (left\_frame): Займає 40% ширини вікна, де розміщені кнопки та поля для взаємодії користувача.

Правий фрейм (right\_frame): Займає 60% ширини вікна, де відображається вибране зображення.

В лівій частині містяться такі копки:

1. Кнопка вибору зображення

2. Вибір мови зі списку

3. Кнопка чекбоксу для випадку коли потрібно розпізнати вертикальний текст

4. Кнопка для виконання розпізнавання

5. Поле, куди буде виводитись розпізнаний текст

# 4. Визначення якості розпізнавання

Більшість зображень будуть зображеннями з книг. Кількість символів на зображеннях різна. Є зображення як і до 100 символів, так і більше ніж 500. Зображення не найкращої якості, багато без фокусу, також на багатьох є затемнення та засвітлення для перевірки як програма буде поратись з такими зображеннями. Звісно, де якість була краща виявся кращим і результат.

Таблиця результатів ймовірності розпізнавання для трьох мов:

Табл.1. Результати розпізнавання

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| jp | name | % | en | name | % | ua | name | % |
| 1 | 20240927\_125032.jpg | 96.41 | 1 | 20241110\_161351.jpg | 99 | 1 | 20241110\_164658.jpg | 99.84 |
| 2 | 20241110\_154428.jpg | 95.98 | 2 | 20241110\_163010.jpg | 99.80 | 2 | 20241110\_164729.jpg | 99.82 |
| 3 | 20241110\_154622.jpg | 87.90 | 3 | 20241110\_161644.jpg | 99.82 | 3 | 20241110\_164400.jpg | 99.42 |
| 4 | 20241110\_154806.jpg | 99.45 | 4 | 20241110\_225730.jpg | 99.85 | 4 | 20241110\_164400.jpg | 94.44 |
| 5 | 20241110\_155426.jpg | 98.84 | 5 | 20241110\_225851.jpg | 99.91 | 5 | 20241110\_164510.jpg | 99.91 |
| 6 | 20241110\_160323.jpg | 100 | 6 | 20241110\_225955.jpg | 99.22 | 6 | 20241110\_164544.jpg | 96.81 |
| 7 | 20241110\_160458.jpg | 98.28 | 7 | 20241110\_230258.jpg | 100 | 7 | 20241110\_163448.jpg | 98.41 |
| 8 | 20241110\_160648.jpg | 83.39 | 8 | 20241110\_230424.jpg | 99.44 | 8 | 20241110\_235732.jpg | 99.44 |
| 9 | 20241110\_211505.jpg | 95.02 | 9 | 20241110\_230455.jpg | 99.63 | 9 | 20241110\_235756.jpg | 87.09 |
| 10 | 20241110\_211505.jpg | 97.42 | 10 | 20241110\_230613.jpg | 99.81 | 10 | 20241110\_235850.jpg | 99.05 |
|  |  | 95.27 |  |  | 99.65 |  |  | 97.42 |

Як можна бачити з результату, середня відсоток розпізнавання серед троьох мов, найбільш високим є в англійської мови. Відчутний вплив дала функція видалення тіней, покращення від 1.5% до 4.6% відсотків.

# Висновки

В результаті виконання курсової роботи було проаналізовано основні характеристики зображення та деякі методи їхньої обробки, алгоритми розпізнавання, що використовуються для символів на зображенні а також деякі відомі готові програмні засоби.

Було розроблено програмний засіб для розпізнавання тексту на зображеннях формату JPEG та PNG з розміром до 5000x5000 пікселів і об'ємом до 5 МБ. Програма має графічний інтерфейс для зручнішого користування. Перед розпізнаванням тексту програмний засіб виконує ряд етапів обробки зображення для підвищення точності розпізнавання. Зокрема, з зображення видаляються тіні, перетворюється до градацій сірого, після чого застосовується фільтрація шумів, покращення контрасту та бінаризація, що робить текст на зображенні чіткішим. Програма також враховує можливість розпізнавання вертикального тексту. Швидкість оброблення та розпізнавання тексту була в межах від 1 до 3 секунд в залежності від якості зображення та середня швидкість близько 2 секунд.

Для тестування зображення було використано 30 зображень, 10 для кожної мови. В результаті для всіх мов середня ймовірність розпізнавання вийшла більше ніж 95%, а середня серед всіх зображень – 97.5%.

# Список літератури.

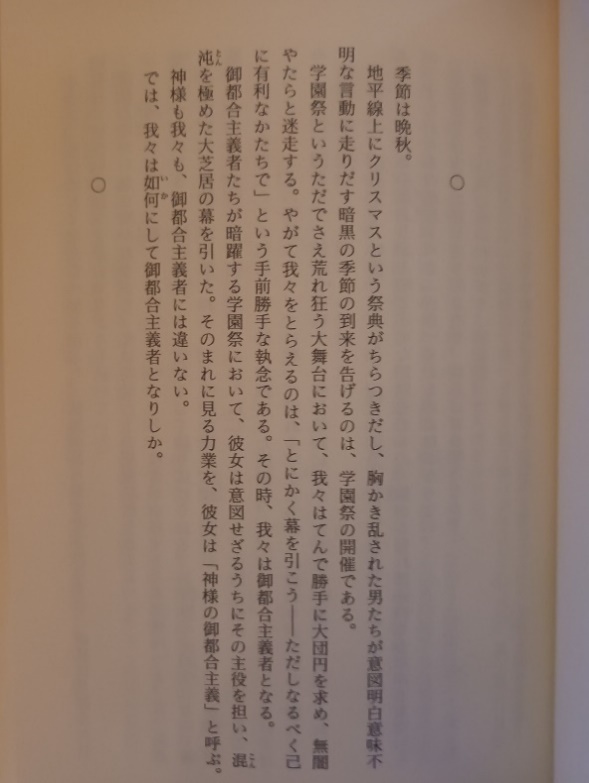
1. Biró A., et al. Synthetized Multilanguage OCR Using CRNN and SVTR Models for Realtime Collaborative Tools. Applied Sciences, 2023. Т. 13, № 7. С. 24. URL: <https://doi.org/10.3390/app13074419> (дата звернення: 07.11.2024).
2. Capture2Text. Capture2Text. URL: <https://capture2text.sourceforge.net/> (дата звернення: 08.11.2024).
3. Cloud Vision API documentation | Google Cloud. Google Cloud. URL: <https://cloud.google.com/vision/docs> (дата звернення: 08.11.2024).
4. Connected Component Analysis in Image Processing - Scaler Topics. Scaler Topics. URL: <https://www.scaler.com/topics/connected-component-analysis-in-image-processing/> (дата звернення: 07.11.2024).
5. Contrast Limited Adaptive Histogram Equalization - MATLAB & Simulink. MathWorks - Maker of MATLAB and Simulink - MATLAB & Simulink. URL: <https://www.mathworks.com/help/visionhdl/ug/contrast-adaptive-histogram-equalization.html> (дата звернення: 08.11.2024).
6. DeepAI. Long Short-Term Memory. DeepAI. URL: <https://deepai.org/machine-learning-glossary-and-terms/long-short-term-memory> (дата звернення: 07.11.2024).
7. Exploring How Gamma Correction Improves Image Accuracy and Contrast. URL: <https://www.displaycalibration.de/en/exploring-how-gamma-correction-improves-image-accuracy-and-contrast> (дата звернення: 07.11.2024).
8. Gonzalez S. Unveiling the Magic of Gaussian Blur: A Smoother World of Images. Salvatore Labs. URL: <https://blog.salvatorelabs.com/unveiling-the-magic-of-gaussian-blur-a-smoother-world-of-images/> (дата звернення: 08.11.2024).
9. Hirata N. S. T., Papakostas G. A. On Machine-Learning Morphological Image Operators. Mathematics. 2021. Т. 9, № 16. С. 22. URL: <https://doi.org/10.3390/math9161854> (дата звернення: 09.11.2024).
10. How to Optimize Image Quality for Better OCR Results. URL: <https://optiic.dev/blog/how-to-optimize-image-quality-for-better-ocr-results> (дата звернення: 07.11.2024).
11. LearnOpenCV – Otsu’s Thresholding Technique. LearnOpenCV. URL: <https://learnopencv.com/otsu-thresholding-with-opencv/> (дата звернення: 08.11.2024).
12. Li M., et al. TrOCR: Transformer-Based Optical Character Recognition with Pre-trained Models. Proceedings of the AAAI Conference on Artificial Intelligence, 2023. Т. 37, № 11. С. 13094–13102. URL: <https://doi.org/10.1609/aaai.v37i11.26538> (дата звернення: 07.11.2024).
13. manga-ocr. PyPI. URL: <https://pypi.org/project/manga-ocr/0.1.8/> (дата звернення: 08.11.2024).
14. Morphological operations – Introduction to Bioimage Analysis. Introduction to Bioimage Analysis – Introduction to Bioimage Analysis. URL: https://bioimagebook.github.io/chapters/2-processing/5-morph/morph.html (дата звернення: 09.11.2024).
15. OpenCV: OpenCV modules. OpenCV documentation index. URL: <https://docs.opencv.org/4.x/index.html> (дата звернення: 08.11.2024).
16. Open CV: How does the BGR2GRAY function work?. Stack Overflow. URL: <https://stackoverflow.com/questions/61058335/open-cv-how-does-the-bgr2gray-function-work> (дата звернення: 07.11.2024).
17. Optimum Global Thresholding using Otsu’s Method. TheAILearner. URL: <https://theailearner.com/2019/07/19/optimum-global-thresholding-using-otsus-method/> (дата звернення: 09.11.2024).
18. Otsu's method. URL: [https://ja.wikipedia.org/wiki/大津の二値化法](https://ja.wikipedia.org/wiki/%E5%A4%A7%E6%B4%A5%E3%81%AE%E4%BA%8C%E5%80%A4%E5%8C%96%E6%B3%95) (дата звернення: 09.11.2024).
19. Patel J. Handwritten And Printed Text Recognition Using Tesseract-OCR. International Journal of Creative Research Thoughts, 2021. Т. 9, № 9. С. 9.
20. Recommended Scan Settings for the Best OCR Accuracy. URL: <https://www.dynamsoft.com/blog/insights/scan-settings-for-best-ocr-accuracy/> (дата звернення: 07.11.2024).
21. Sharda A. Image Filters: Gaussian Blur. Medium. URL: <https://aryamansharda.medium.com/image-filters-gaussian-blur-eb36db6781b1> (дата звернення: 08.11.2024).
22. Taye M. M. Theoretical Understanding of Convolutional Neural Network: Concepts, Architectures, Applications, Future Directions. Computation, 2023. Т. 11, № 3. С. 52. URL: <https://doi.org/10.3390/computation11030052> (дата звернення: 07.11.2024).
23. Techniques and Challenges of Image Segmentation: A Review / Y. Yu et al. Electronics, 2023. Т. 12, № 5. С. 24. URL: <https://doi.org/10.3390/electronics12051199> (дата звернення: 07.11.2024).
24. Tesseract OCR vs. CNN-based OCR: Which is Right for You?. URL: <https://www.veryfi.com/technology/tesseract-ocr-vs-cnn-based-ocr/> (дата звернення: 07.11.2024).
25. Three algorithms for converting color to grayscale. Grayscale Images. URL: <https://www.grayscaleimage.com/three-algorithms-for-converting-color-to-grayscale/> (дата звернення: 08.11.2024).
26. Vision | Apple Developer Documentation. Apple Developer Documentation. URL: <https://developer.apple.com/documentation/vision/> (дата звернення: 08.11.2024).
27. Wennerberg J. Automating template generation for pattern matching OCR systems. Jönköping, 2024. 90 с.

# Додатки

## Додаток А

import cv2  
import pytesseract  
import numpy as np  
import tkinter as tk  
from tkinter import filedialog, messagebox, ttk, simpledialog  
from PIL import Image, ImageTk, ExifTags  
import os  
import time  
from difflib import SequenceMatcher  
import re  
  
pytesseract.pytesseract.tesseract\_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe' *# Потрібно вказати шлях*def clean\_text\_japanese(text):  
 *"""Прибирає зайві символи, пробіли та з'єднує фрагментовані слова в японському тексті."""* cleaned\_text = re.sub(r'[^a-zA-Z0-9\u3040-\u309f\u30a0-\u30ff\u4e00-\u9fff\s\.,!?\'"()-]', '', text) cleaned\_text = re.sub(r'\s\*([、。！？.,!?])\s\*', r'\1', cleaned\_text) cleaned\_text = re.sub(r'(?<=\w)\s+(?=\w)', '', cleaned\_text) cleaned\_text = re.sub(r'(?<![\u30a0-\u30ff])ー(?![\u30a0-\u30ff])', '', cleaned\_text) cleaned\_text = re.sub(r'([ー]{2,})', 'ー', cleaned\_text)  
 cleaned\_text = re.sub(r'\s+', ' ', cleaned\_text).strip()return cleaned\_text  
  
def clean\_text\_english(text):  
 *"""Очищає зайві символи, об’єднує розірвані слова для англійського тексту,  
 додає пробіли після знаків пунктуації та видаляє зайві пробіли."""* cleaned\_text = re.sub(r'[^a-zA-Z0-9\s\.,!?\'"-]', '', text)cleaned\_text = re.sub(r'(?<=\w)\s+(?=\w)', ' ', cleaned\_text)cleaned\_text = re.sub(r'([.,!?])(?=\S)', r'\1 ', cleaned\_text)cleaned\_text = re.sub(r'\s\*([.,!?])\s\*', r'\1 ', cleaned\_text)cleaned\_text = re.sub(r'\s+', ' ', cleaned\_text).strip()  
 return cleaned\_text  
  
def clean\_text\_ukrainian(text):  
 *"""Очищує зайві символи, об’єднує розірвані слова для українського тексту,  
 додає пробіли після знаків пунктуації та видаляє зайві пробіли."""* cleaned\_text = re.sub(r'[^a-zA-Zа-яА-ЯіІїЇєЄ0-9\s\.,!?\'"-]', '', text)cleaned\_text = re.sub(r'(?<=\w)\s+(?=\w)', ' ', cleaned\_text)cleaned\_text = re.sub(r'([.,!?])(?=\S)', r'\1 ', cleaned\_text)cleaned\_text = re.sub(r'\s\*([.,!?])\s\*', r'\1 ', cleaned\_text)cleaned\_text = re.sub(r'\s+', ' ', cleaned\_text).strip()  
  
 return cleaned\_text  
  
def clean\_text(text, language):  
 if language == 'jpn':  
 return clean\_text\_japanese(text)  
 elif language == 'eng':  
 return clean\_text\_english(text)  
 elif language == 'ukr':  
 return clean\_text\_ukrainian(text)  
 else:  
 return text  
  
def calculate\_similarity(reference\_text, recognized\_text):matcher = SequenceMatcher(None, reference\_text, recognized\_text)correct\_chars = 0for tag, i1, i2, j1, j2 in matcher.get\_opcodes():  
 if tag == 'equal':  
 correct\_chars += (i2 - i1) elif tag == 'replace':correct\_chars += min(i2 - i1, j2 - j1)total\_chars = len(reference\_text)if total\_chars == 0:  
 return 0.0accuracy\_percentage = (correct\_chars / total\_chars) \* 100  
 return accuracy\_percentage  
  
def check\_image\_size(image\_path):file\_size = os.path.getsize(image\_path)  
 if file\_size > 5 \* 1024 \* 1024: *# 5 МБ* return False  
 return True  
  
def adjust\_gamma(image, gamma=1.0):  
 *"""Функція корекції гамми."""* inv\_gamma = 1.0 / gamma  
 table = np.array([((i / 255.0) \*\* inv\_gamma) \* 255 for i in np.arange(0, 256)]).astype("uint8")  
 return cv2.LUT(image, table)  
  
  
def remove\_shadows(image):  
 *"""Функція видалення тіней з зображення."""* rgb\_planes = cv2.split(image) *# Розділити канали RGB* result\_planes = []  
 for plane in rgb\_planes:  
 *# Використання морфологічного закриття для оцінки фону* dilated\_img = cv2.dilate(plane, np.ones((7, 7), np.uint8))  
 bg\_img = cv2.medianBlur(dilated\_img, 21)  
 *# Віднімання фону від оригінального зображення* diff\_img = 255 - cv2.absdiff(plane, bg\_img)  
 result\_planes.append(diff\_img)  
  
 result = cv2.merge(result\_planes) *# Об'єднати оброблені канали* return result  
  
def ocr\_from\_image(image\_path, language='ukr'):  
 *"""Основна функція обробки та розпізнавання тексту."""* image = cv2.imread(image\_path) *# Завантаження зображення* shadow\_removed = remove\_shadows(image) *# Видалення тіней* gray = cv2.cvtColor(shadow\_removed, cv2.COLOR\_BGR2GRAY) *# Перетворення в градації сірого* gamma\_corrected = adjust\_gamma(gray, gamma=4) *# Корекція гамми для зменшення засвітлення* clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8, 8)) *# Застосування CLAHE для покращення контрасту* enhanced\_image = clahe.apply(gamma\_corrected)  
 blurred = cv2.GaussianBlur(enhanced\_image, (3, 3), 0) *# Розмивання зображення (щоб зменшити шум)* \_, binary = cv2.threshold(blurred, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU) *# Бінаризація зображення для чіткості тексту* kernel = np.ones((1, 1), np.uint8) *# Застосування морфологічних операцій для покращення текстових об'єктів* dilated = cv2.dilate(binary, kernel, iterations=1)  
 eroded = cv2.erode(dilated, kernel, iterations=1)  
  
 cv2.imwrite('processed\_image.png', eroded)  
 if is\_vertical and language == 'jpn':  
 custom\_config = f'--oem 3 --psm 5 -l jpn\_vert -c preserve\_interword\_spaces=1'  
 else:  
 custom\_config = f'--oem 3 --psm 6 -l {language}'  
 text = pytesseract.image\_to\_string(eroded, config=custom\_config)  
  
 return text  
  
def select\_image():  
 file\_path = filedialog.askopenfilename(filetypes=[("Image files", "\*.jpg;\*.jpeg;\*.png")])  
 if file\_path:  
 if not check\_image\_size(file\_path):  
 messagebox.showwarning("Warning", "Image size exceeds 5 MB.")  
 return  
 image\_path.set(file\_path)  
 load\_image(file\_path)  
  
def load\_image(image\_path):img = Image.open(image\_path)try:  
 for orientation in ExifTags.TAGS.keys():  
 if ExifTags.TAGS[orientation] == 'Orientation':  
 break  
 exif = img.\_getexif()  
 if exif is not None:  
 orientation\_value = exif.get(orientation, None)  
 if orientation\_value == 3:  
 img = img.rotate(180, expand=True)  
 elif orientation\_value == 6:  
 img = img.rotate(270, expand=True)  
 elif orientation\_value == 8:  
 img = img.rotate(90, expand=True)  
 except (AttributeError, KeyError, IndexError):  
 pass  
  
 available\_width = int(root.winfo\_width() \* 0.6)  
 available\_height = int(root.winfo\_height() \* 0.85)  
  
 img\_width, img\_height = img.size  
 ratio = min(available\_width / img\_width, available\_height / img\_height)  
 new\_width = int(img\_width \* ratio)  
 new\_height = int(img\_height \* ratio)  
  
 img = img.resize((new\_width, new\_height), Image.Resampling.LANCZOS)  
 img\_tk = ImageTk.PhotoImage(img)  
 image\_label.config(image=img\_tk)  
 image\_label.image = img\_tk  
  
  
def perform\_ocr():  
 if not image\_path.get():  
 messagebox.showwarning("Warning", "Please select an image.")  
 return  
 start\_time = time.time()  
 language = languages[language\_combobox.get()]  
 recognized\_text = ocr\_from\_image(image\_path.get(), language)  
 recognized\_text = clean\_text(recognized\_text, language)end\_time = time.time()  
 processing\_time = end\_time - start\_time  
 recognized\_text\_box.delete(1.0, tk.END)  
 recognized\_text\_box.insert(tk.END, recognized\_text)  
 print(f"Time taken for OCR: {processing\_time:.2f} seconds")reference\_text = simpledialog.askstring("Reference Text", "Enter the reference text for comparison:")  
 if reference\_text:accuracy = calculate\_similarity(reference\_text, recognized\_text)  
 messagebox.showinfo("OCR Accuracy", f"Accuracy of recognition: {accuracy:.2f}%")with open("recognized\_text.txt", "w", encoding='utf-8') as f:  
 f.write(recognized\_text)  
  
is\_vertical = False  
  
def toggle\_vertical\_text():global is\_vertical  
 is\_vertical = not is\_vertical  
 if is\_vertical:  
 vertical\_status\_label.config(text="Vertical Text Mode: Enabled", foreground='green')  
 else:  
 vertical\_status\_label.config(text="Vertical Text Mode: Disabled", foreground='red')  
  
root = tk.Tk()  
root.title("📄 OCR Text Recognition")  
root.geometry(f"{root.winfo\_screenwidth()}x{root.winfo\_screenheight()}")  
root.state('zoomed')  
root.configure(bg='#2C3E50')  
  
image\_path = tk.StringVar()  
  
languages = {  
 'English': 'eng',  
 'Ukrainian': 'ukr',  
 'Japanese': 'jpn',  
}  
  
left\_frame = ttk.Frame(root, padding=20, style='TFrame')  
left\_frame.place(relx=0, rely=0, relwidth=0.4, relheight=1)  
  
right\_frame = ttk.Frame(root, padding=20, style='TFrame')  
right\_frame.place(relx=0.4, rely=0, relwidth=0.6, relheight=1)  
  
select\_button = ttk.Button(  
 left\_frame, text="🖼 Select Image", command=select\_image, style='Accent.TButton'  
)  
select\_button.pack(pady=20)  
  
language\_label = ttk.Label(left\_frame, text="Select Language:", font=('Helvetica', 14, 'bold'), background='#2C3E50', foreground='white')  
language\_label.pack(pady=10)  
  
language\_combobox = ttk.Combobox(left\_frame, values=list(languages.keys()), font=('Helvetica', 12))  
language\_combobox.set("English")  
language\_combobox.pack(pady=10)  
  
*# Add a checkbox to toggle vertical text recognition*vertical\_text\_var = tk.BooleanVar() *# Variable to store the state of the checkbox*vertical\_checkbox = ttk.Checkbutton(  
 left\_frame, text="Enable Vertical Text", variable=vertical\_text\_var, command=toggle\_vertical\_text,  
 style='Accent.TButton'  
)  
vertical\_checkbox.pack(pady=10)  
vertical\_status\_label = ttk.Label(left\_frame, text="Vertical Text Mode: Disabled", font=('Helvetica', 12), background='#2C3E50', foreground='red')  
vertical\_status\_label.pack(pady=10)  
  
ocr\_button = ttk.Button(  
 left\_frame, text="🔍 Perform OCR", command=perform\_ocr, style='Accent.TButton'  
)  
ocr\_button.pack(pady=20)  
recognized\_text\_label = ttk.Label(left\_frame, text="Recognized Text:", font=('Helvetica', 14, 'bold'), background='#2C3E50', foreground='white')  
recognized\_text\_label.pack(pady=10)  
  
recognized\_text\_box = tk.Text(left\_frame, wrap='word', font=('Helvetica', 12), height=15, width=40, background='#34495E', foreground='white')  
recognized\_text\_box.pack(pady=10, fill='both', expand=True)  
  
image\_label = ttk.Label(right\_frame, background='#34495E', foreground='#2C3E50', font=('Helvetica', 10), anchor='center')  
image\_label.pack(pady=10, fill='both', expand=True)  
  
style = ttk.Style()  
style.theme\_use('clam')  
style.configure('TFrame', background='#2C3E50')  
style.configure('TLabel', background='#2C3E50', foreground='white', font=('Helvetica', 12))  
style.configure('Accent.TButton', background='#3498DB', foreground='white', font=('Helvetica', 14, 'bold'),  
 borderwidth=0)  
style.map('Accent.TButton', background=[('active', '#2980B9')])  
  
root.mainloop()

## Додаток Б



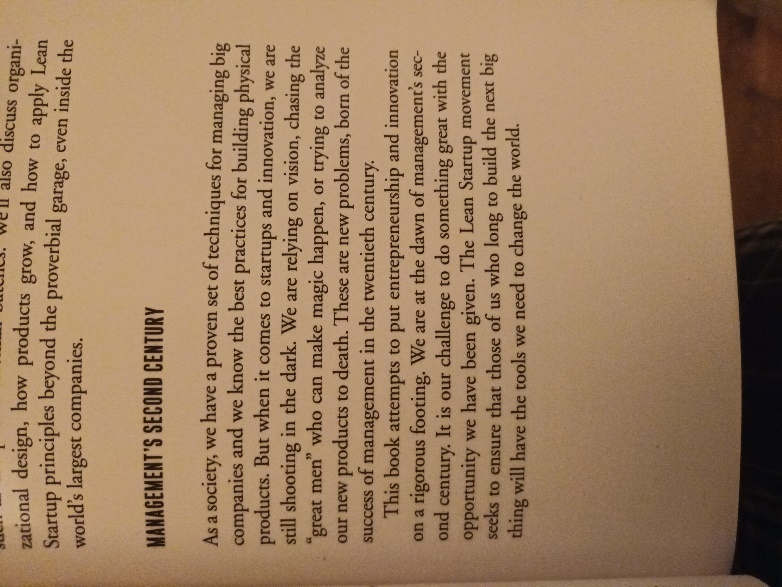
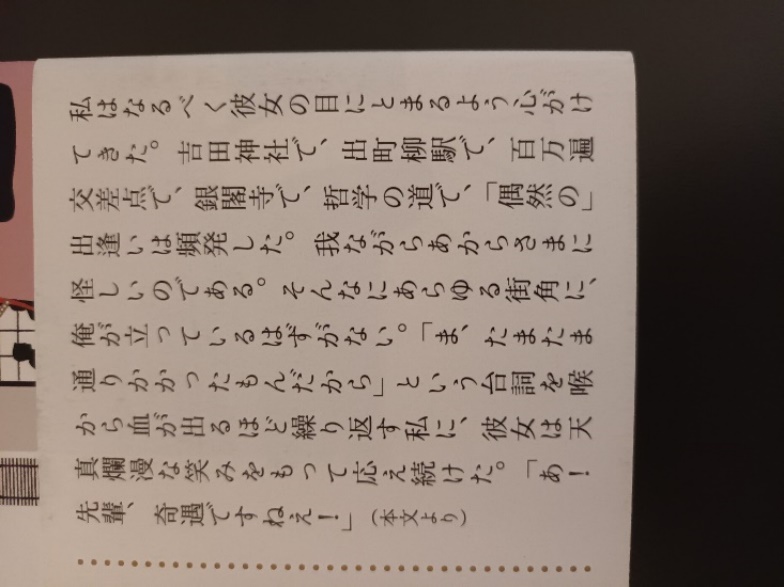
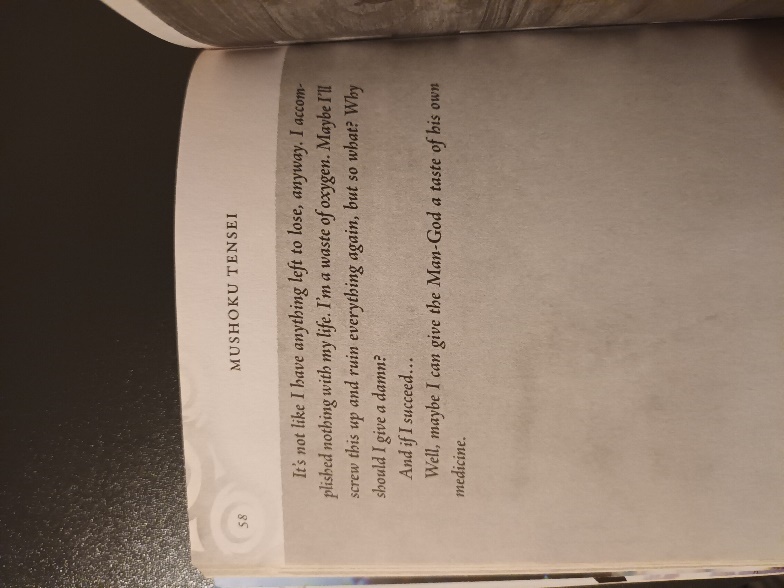


Рис. Б.4 Зображення з найменшим відсотком розпізнавання англійською

Рис. Б.2 Зображення з найменшим відсотком розпізнавання японською

Рис. Б.1 Зображення з найбільшим відсотком розпізнавання японською

Рис. Б.3 Зображення з найбільшим відсотком розпізнавання англійською

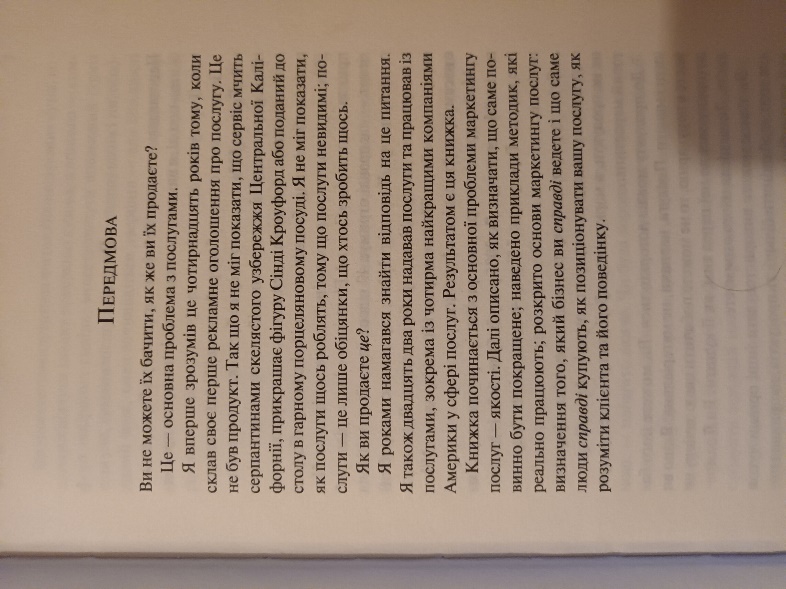
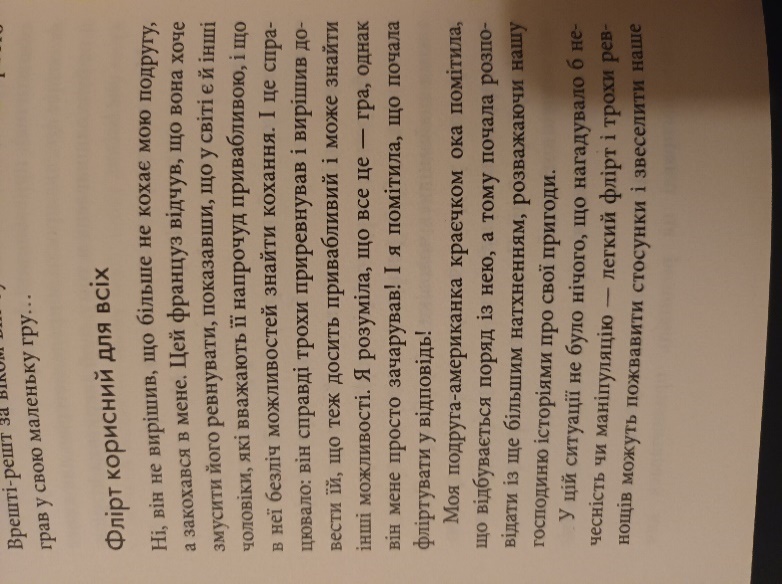


Рис. Б.5 Зображення з найбільшим відсотком розпізнавання українською

Рис. Б.6 Зображення з найменшим відсотком розпізнавання українською