МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»

ОБ’ЄКТНО-ОРІЄНТОВАНЕ ПРОЄКТУВАННЯ ТЕХНІЧНИХ СИСТЕМ

Частина 2

СТВОРЕННЯ ТА ВИКОРИСТАННЯ СИСТЕМ ОПТИЧНОГО РОЗПІЗНАВАННЯ ТЕКСТІВ

Навчальний посібник

Харків «ХАІ» 2024

УДК 004.93’1(075.8)
О12

Колектив авторів:
Л. О. Краснов, В. О. Білозерський, К. Ю. Дергачов, А. Я. Зимовін

Рецензенти: д-р техн. наук, проф. В. П. Лисечко,
канд. техн. наук С. М. Флерко

Об’єктно-орієнтоване проєктування технічних систем [Електронний ресурс] : навч. посіб. В 2 ч. Створення та використання систем оптичного розпізнавання текстів / Л. О. Краснов, В. О. Білозерський, К. Ю. Дергачов, А. Я. Зимовін. – Харків : Нац. аерокосм. ун-т ім. М. Є. Жуковського «Харків. авіац. ін-т», 2024. – 116 с.

О12

Подано матеріал для практичного вивчення методів побудови, оптимізації та експлуатації сучасних систем оптичного розпізнавання текстів. Запропоновано сучасну концепцію побудови систем розпізнавання текстів, яка враховує необхідність попереднього оброблення вихідних даних для компенсації негативних зовнішніх впливів. Розглянуто набір сервісних функцій для перегляду, редагування та збереження текстів у стандартних форматах, а також заходи щодо захисту конфіденційної інформації під час її передавання відкритими каналами зв'язку.

Для студентів, що навчаються за напрямами підготовки «Авіоніка», «Аеронавігація» і «Системна інженерія» спеціальностей «Системи керування літальними апаратами та комплексами», «Комп`ютеризовані системи управління та автоматика», «Аеронавігаційні системи і системи аеронавігаційного обслуговування».

Іл. 111. Табл. 7. Бібліогр.: 29 назв.

УДК 004.93’1(075.8)

ВСТУП

Нині розвиток інформаційних технологій створює об'єктивні передумови для успішного розв’язання фундаментальних проблем розпізнавання образів. Сучасники є свідками й учасниками процесу стрімкого зростання цифрових технологій на основі алгоритмів штучного інтелекту. Бурхливий розвиток комп'ютерних систем (істотне підвищення швидкодії, значне збільшення обсягів пам'яті, створення нових операційних систем і мов програмування) природно привели до масового впровадження інноваційних методів оброблення даних. Особливо ця тенденція помітна у сфері комп'ютерного зору під час вирішення широкого кола завдань розпізнавання образів і побудови сучасних систем технічного зору (СТЗ).

Найбільш важливими й актуальними в межах глобальної проблеми розпізнавання образів є дві групи завдань – виявлення та розпізнавання осіб (face detection and recognition), а також завдання оптичного розпізнавання тексту на зображеннях (optical character recognition, OCR) для подальшого перетворення на електронний формат. Практичні потреби у вирішенні таких завдань постійно зростають.

Значних успіхів під час вирішення завдань розпізнавання та класифікації об'єктів досягнуто завдяки застосуванню сучасних методів та алгоритмів побудови й навчання глибинних нейронних мереж DNN (Deep Neural Network), серед яких найбільш важливу роль відіграють згорткові нейронні мережі CNN (Convolutional Neural Network). Однак при цьому виникає низка важливих і складних проблем, пов'язаних із різними обмеженнями, як-от: негативний вплив на процес оброблення зображень низкої якості фото- та відеознімання, недостатнє освітлення сцени, геометричні спотворення та інші фактори. У зв'язку із цим потрібні попередні етапи оброблення та підготовки даних для їх успішного розпізнавання та класифікації. На жаль, нині немає універсальних рекомендацій, а до того ж стандартів у цій галузі.

Слід зазначити, що в сучасних системах оптичного розпізнавання тексту недостатньо розвинені сервісні функції, пов'язані з переглядом і збереженням отриманих текстових даних. Майже не приділяється уваги проблемі захисту інформації в ситуаціях, коли ці дані мають конфіденційний характер. Особливо цей недолік стає помітним тоді, коли необхідно передати дані відкритими каналами зв'язку.

Усе це потребує продовження досліджень, розроблення та впровадження нових, ефективніших алгоритмів і методик, створення інноваційних систем розпізнавання тексту, впровадження їх у повсякденну практику під час використання інформаційних технологій.

1. АНАЛІЗ РЕСУРСІВ ДЛЯ СТВОРЕННЯ ІНОВАЦІЙНИХ СИСТЕМ OCR

У цьому розділі запропоновано результати комплексу досліджень щодо створення систем оптичного розпізнавання тексту на зображеннях із подальшим перетворенням результатів розпізнавання на електронний текстовий формат. Дослідження проводились із використанням найпопулярніших інноваційних технологій. Кінцевою метою було завдання щодо створення сучасної системи конвертації текстових документів із паперового в електронний формат. Ця система має забезпечувати високу якість розпізнавання тексту, незважаючи на різноманітні фактори та впливи, що перешкоджають, містити набір розвинених сервісних функцій і забезпечувати ефективний захист інформації під час передавання її каналами зв'язку.

1.1. Цілі й основні завдання досліджень

Було сформульовано завдання проведення досліджень у таких основних напрямах:

- вирішити поставлене завдання розпізнавання тексту на базі нейромережевих технологій, використовуючи знімки паперових документів, отриманих за допомогою будь-яких доступних фотореєстраторів (фотоапаратів, телефонів, вебкамер тощо);
- дослідити сучасні алгоритми попереднього оброблення вихідних зображень і створити на їх основі пакет методик, що дають змогу компенсувати негативний вплив зовнішніх факторів на якість роботи системи (низька якість фотознімків, погані умови освітлення під час фотографування, несприятливий геометричний фактор та ін.);
- у системі створити набір сервісних функцій, які забезпечують можливість перегляду та редагування поточної інформації, її збереження в стандартних форматах;
- розробити технологію та алгоритми кодування текстової інформації, отриманої після розпізнавання, для передавання її відкритими каналами зв'язку зі збереженням конфіденційності.

1.2. Базова концепція побудови системи розпізнавання текстів

Для успішної реалізації проєкту було сформульовано розгорнуту концепцію побудови системи оптичного розпізнавання тексту, що дає змогу оптимізувати зміст і порядок виконання етапів. Основні положення й етапи цієї концепції такі:

- докланий аналіз сучасного стану нейромережевих технологій, оцінювання їх переваг і недоліків під час вирішення завдання оптичного розпізнавання тексту;
- на основі порівняльного аналізу вибір найбільш перспективної системи оптичного розпізнавання тексту, яку доцільно використовувати як інформаційне ядро проєкту;

- визначення складу основних інструментальних і програмних ресурсів, необхідних для реалізації проєкту;
- оцінювання основних зовнішніх чинників негативного впливу на роботу системи розпізнавання текстів;
- формулювання критерію та показників оцінювання якості розпізнавання тексту;
- створення набору методик попереднього оброблення зображень, що дає змогу ефективно компенсувати негативний вплив зовнішніх факторів;
- синтез сервісних функцій для зручності перегляду даних, настроювання параметрів системи, збереження результатів роботи в стандартних форматах;
- розроблення методів і алгоритмів захисту інформації в мережі «Інтернет» у процесі передавання конфіденційних даних, отриманих під часі розпізнавання текстових документів;
- створення та тестування програмного забезпечення для практичної реалізації результатів досліджень.

1.3. Аналіз сучасних нейромережевих ресурсів

Постійне збільшення потреб у вирішенні за допомогою СТЗ завдань виявлення різноманітних об'єктів, їх розпізнавання і класифікації сприяє впровадженню все новіших і прогресивніших технічних рішень за допомогою нейронних мереж. Стрімко змінюється архітектура нейронних мереж, удосконалюються методи їх навчання, з'являються нові методи керування наявними ресурсами, створюються спеціалізовані бібліотеки. У зв'язку із цим доцільно провести аналіз сучасних методів керування наявними ресурсами підтримки нейронних мереж глибокого навчання за допомогою бібліотеки OpenCV.

Відомо, що нейронні мережі є одним із різновидів алгоритмів машинного навчання (machine learning). Основною властивістю алгоритмів machine learning є їх здатність навчатися в процесі роботи. Наприклад, під час побудови алгоритму «дерева рішень» немає апріорної інформації про характер вхідних даних. Використовується тільки певний вхідний набір об'єктів і значення ознак для кожного з них разом із міткою класу. У процесі побудови «дерева рішень» алгоритм сам виявляє приховані закономірності, тобто навчається, і після навчання здатний передбачати клас уже для нових об'єктів, які він не бачив раніше.

Глибоке навчання (deep learning) в теорії машинного навчання є підмножиною так званого representation learning. Основна концепція representation learning − автоматичний пошук ознак, за якими в подальшому буде працювати певний алгоритм, наприклад класифікації.

Ще одна важлива проблема під час використання машинного навчання − наявність факторів мінливості зображень. Це істотно впливає на вигляд вихідних даних. У завданнях розпізнавання зображень такими факто-

рами можуть бути кут, під яким предмет на зображенні повернений до спостерігача, освітлення тощо. Тому для завдань виявлення й ідентифікації об'єктів розумно враховувати не конкретні низькорівневі факти, такі як колір певного пікселя, а характеристики вищого рівня абстракції. Однак через наявність тільки одного або декількох можливих ознак процеси визначення всіх можливих ознак і складання алгоритмів для перевірки зображення на наявність цих ознак є малопродуктивними. У таких ситуаціях краще використовувати переваги підходу deep learning.

Одним із найбільш ефективних методів вирішення завдань розпізнавання образів є використання згорткових нейронних мереж CNN (Convolutional Neural Network). Це основний інструмент для класифікації та розпізнавання зображень і сигналів. Є безліч варіантів застосування CNN, як-от: Deep Convolutional Neural Network (DCNN), Region-CNN (R-CNN), Fully Convolutional Neural Networks (FCNN) та ін.

Одна з перших робіт, присвячених новій архітектурі згорткових нейронних мереж [1], була опублікована в 1998 році й отримала назву CNN (Convolutional Neural Network). Довгий час така мережа залишалася поза увагою, однак у 2012 році після змагань у галузі комп'ютерного зору, що проводяться в межах проєкту ILSVRC (ImageNet Large Scale Visual Recognition Challenge − кампанія з широкомасштабного розпізнавання образів на основі бази даних ImageNet [2]), була розроблена згорткова нейронна мережа AlexNet [3]. Ця мережа здатна класифікувати мільйони зображень із тисяч різних категорій, помиляючись лише на 15.8 % [4]. Мережа AlexNet стала переможницею LSVRC-2012, являючи собою просту, але потужну мережеву архітектуру зі згортковими рівнями один поверх іншого. Ця архітектура зазвичай використовується як початок глибокого навчання в завданнях комп'ютерного зору. Ці результати отримали широке визнання розробників і користувачів, зокрема стали стимулом до впровадження CNN у різних додатках із розпізнавання образів у СТЗ.

У зв'язку зі зростанням потреб у застосуванні згорткових нейромережевих технологій докладають значних зусиль і для розвитку відповідних сервісних ресурсів. Це дає змогу істотно зменшити навантаження на розробників ПЗ з навчання нейронних мереж. Прикладом цього може слугувати розширення функціональних можливостей відомої бібліотеки OpenCV, яка часто застосовується для збільшення можливостей мови Python.

Ідея ієрархічного узагальнення й керування наявними ресурсами підтримки нейронних мереж глибокого навчання була реалізована в бібліотеці OpenCV шляхом створення пакета функцій і алгоритмів для керування параметрами мереж глибокого навчання. Для цього, починаючи з версії OpenCV 3.1, у бібліотеку було введено модуль глибоких нейронних мереж (DNN), який реалізує прямий зв'язок із глибокими мережами, попередньо навчений із використанням популярних фреймворків глибокого навчання.

Наведемо приклади найбільш популярних бібліотек для побудови сучасних нейронних мереж.

Бібліотека Caffe [5], розроблена Berkley Vision and Learning Center, сконцентрована на ефективній реалізації алгоритмів глибокого навчання. Як і всі попередні бібліотеки, Caffe має відкритий вихідний код, реалізована мовою C, проте надає також зручний інтерфейс для Python. Підтримує повнозв'язні та згорткові мережі, описує мережі у вигляді набору шарів у форматі `.prototxt`, підтримує обчислення на GPU. До переваг бібліотеки належить також наявність великої кількості попередньо навчених моделей і прикладів, що в поєднанні з іншими характеристиками робить бібліотеку найбільш простою для початку роботи.

Бібліотека TensorFlow [6], розроблена корпорацією Google, призначена для роботи з тензорами, широко використовується для побудови нейронних мереж. Обчислення на відкритих мережах підтримуються мовою програмування C ++. На основі цієї бібліотеки будуються більш високорівневі бібліотеки для роботи з нейронними мережами на рівні цілих шарів. Бібліотека Keras почала використовувати Tensorflow як основний інструмент для обчислень. Також серед унікальних особливостей TensorFlow можна виділити, що основна бібліотека підходить для сімейства техніки машинного навчання, а не тільки для глибинного навчання.

Крім основної функціональності машинного навчання, TensorFlow також має власну систему логування, власний інтерактивний візуалізатор логів і навіть потужну архітектуру доставки даних.

Наукова обчислювальна платформа *Torch / Pytorch* [7] із широкою підтримкою алгоритмів машинного навчання ставить графічні процесори на перше місце. Ця платформа проста у використанні й ефективна завдяки простій і швидкій мові сценаріїв LuaJIT і базовій реалізації C/CUDA.

Короткий огляд основних функцій:

- потужний N-вимірний масив;
- безліч процедур для індексування, нарізання, транспонування;
- зручний і наглядний інтерфейс мовою LuaJIT;
- підпрограми лінійної алгебри;
- нейронні мережі й енергетичні моделі;
- підпрограми чисельної оптимізації;
- швидка й ефективна підтримка графічного процесора;
- можливість вбудовування з портами на серверні системи iOS та Android.

Мета Torch – мати максимальну гнучкість і швидкість у створенні наукових алгоритмів, а також зробити процес надзвичайно простим. Torch, побудована на основі спільноти Lua, постачається з великою екосистемою керованих спільнотою пакетів машинного навчання, комп’ютерного зору, оброблення сигналів, паралельного оброблененя зображень, відео, аудіо та мереж. Фреймворк нейронної мережі Darknet [8] з відкритим вихідним кодом, написаний мовою C і CUDA. Платформа Torch швидка, проста у встановленні та підтримує обчислення CPU та GPU.

Бібліотека Keras [9] для побудови нейронних мереж підтримує основні види шарів і структурні елементи. Підтримує як рекурентні, так і згорткові нейромережі, має у своєму складі реалізацію відомих архітектур нейромереж (наприклад, VGG16). Нещодавно шари із цієї бібліотеки стали доступні всередині бібліотеки Tensorflow. Існують готові функції для роботи із зображеннями і текстом. Ця бібліотека дає змогу на більш високому рівні працювати з нейронними мережами.

Keras зменшує когнітивне навантаження розробника, що дає змогу зосередитися на дійсно важливих проблемах, використовує принцип прогресивного розкриття складності (прості робочі процеси мають бути швидкими та простими, тоді як робочі процеси високого рівня − можливими чітким способом, який ґрунтується на тому, що користувач уже вміє) та забезпечує потужну продуктивність і масштабованість. Саме тому Keras використовують організації та компанії, зокрема NASA, YouTube або Waymo.

У OpenCV 3.3 статус модуля Deep learning був підвищений із репозиторію `opencv_contrib` до основного сховища [10]. У цій версії також було значно збільшено швидкодію модуля DNN.

Основна можливість DNN полягає в завантаженні та запуску нейронних мереж (`inference`). При цьому модель може бути створено в будь-якому з трьох фреймворків глибинного навчання Caffe, TensorFlow або Torch; спосіб її завантаження та використання зберігається незалежно від того, де її було створено.

Підтримуються всі основні шари: від базових (Convolution і Fully connected) до більш спеціалізованих – загалом понад 30. Крім підтримки окремих шарів, важлива також підтримка конкретних архітектур нейронних мереж. Модуль містить приклади для класифікації (AlexNet, GoogLeNet, ResNet, SqueezeNet), сегментації (FCN, ENet), детектування об'єктів (SSD).

Розглянуті бібліотеки та фреймворки використовуються для створення та навчання нейронних мереж із подальшим використанням під час вирішення широкого кола завдань розпізнавання образів, однією з яких є оптичне розпізнавання тексту.

1.4. Порівняльний аналіз систем оптичного розпізнавання тексту

Оптичне розпізнавання тексту (optical character recognition, OCR) − це механічне або електронне перетворення зображень рукописного, машинописного або друкованого тексту в послідовність кодів, що використовуються в текстових редакторах. Розпізнавання широко використовується для конвертації книг і документів в електронний формат (рис. 1.1), для автоматизації систем обліку в бізнесі або для публікації тексту на вебсторінці [11].

Оптичне розпізнавання тексту дає змогу редагувати текст, здійснювати пошук слів або фраз, зберігати результати в компактній формі, демонструвати або роздруковувати матеріал, не втрачаючи якості, аналізувати інформацію, а також застосовувати до тексту електронний переклад, фор-

матування або перетворення в мовлення. Оптичне розпізнавання тексту є досліджуваною проблемою в галузях розпізнавання образів, штучного інтелекту й комп'ютерного зору.

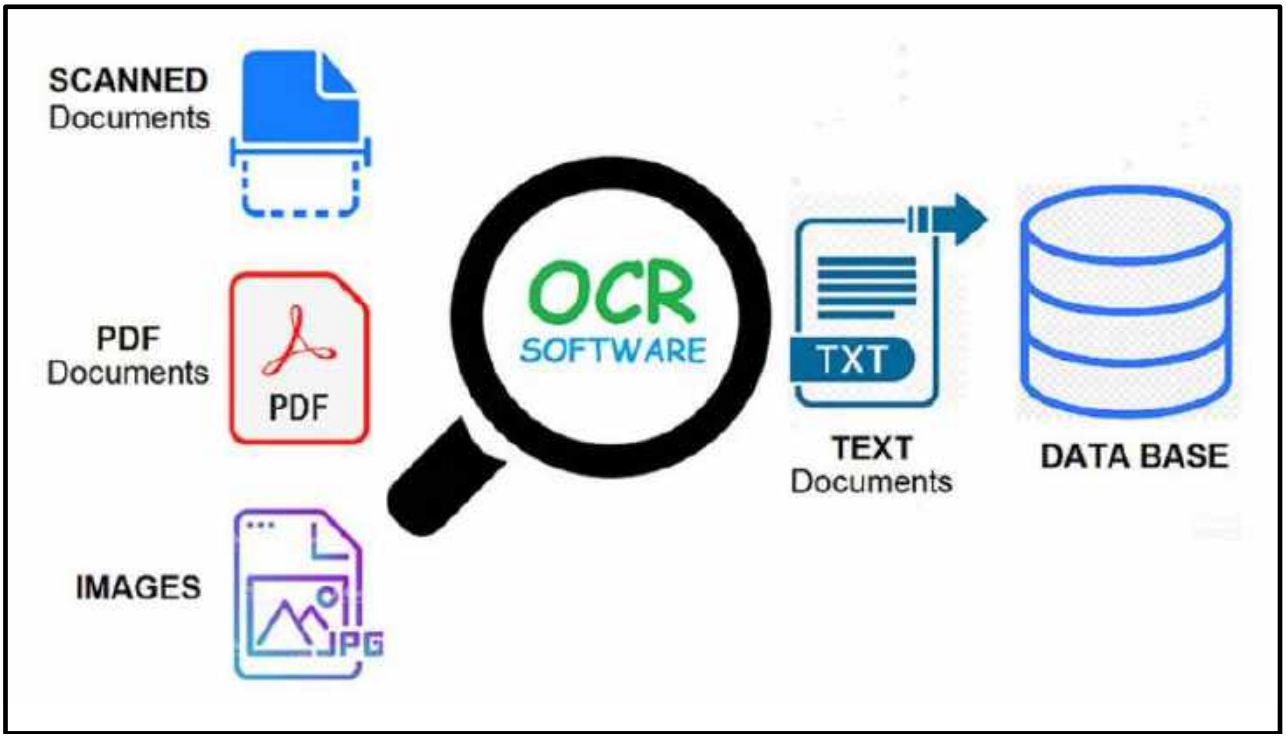


Рис. 1.1. Процес оптичного розпізнавання тексту

Системи оптичного розпізнавання тексту потребують калібрування для роботи з конкретним шрифтом; у ранніх версіях для програмування було необхідно зображення кожного символу, програма одночасно могла працювати тільки з одним шрифтом. Зараз найпоширеніші так звані інтелектуальні системи, що розпізнають більшість шрифтів із високим ступенем точності. Деякі системи оптичного розпізнавання тексту здатні відновлювати вихідне форматування тексту, зокрема зображення, колонки й інші нетекстові компоненти.

Точне розпізнавання латинських символів у друкованому тексті зараз можливе тільки тоді, коли доступні такі чіткі зображення, як друковані документи. Точність у такому разі перевищує 99 %, абсолютної точності можна досягти тільки шляхом подальшого редагування людиною. Проблеми розпізнавання рукописного тексту, а також друкованих текстів інших форматів (особливо з дуже великою кількістю символів) зараз є предметом досліджень [12].

Сьогодні існують готові програмні рішення з оптичного розпізнавання тексту. На ринку IT-послуг є безліч як безкоштовних, так і платних додатків із розпізнавання текстів (рис. 1.2).

Розглянемо лише ті програмні рішення з оптичного розпізнавання текстів, які демонструють хороші результати, визначимо їх переваги і недоліки, а також спробуємо оцінити перспективи їх подальшого розвитку.

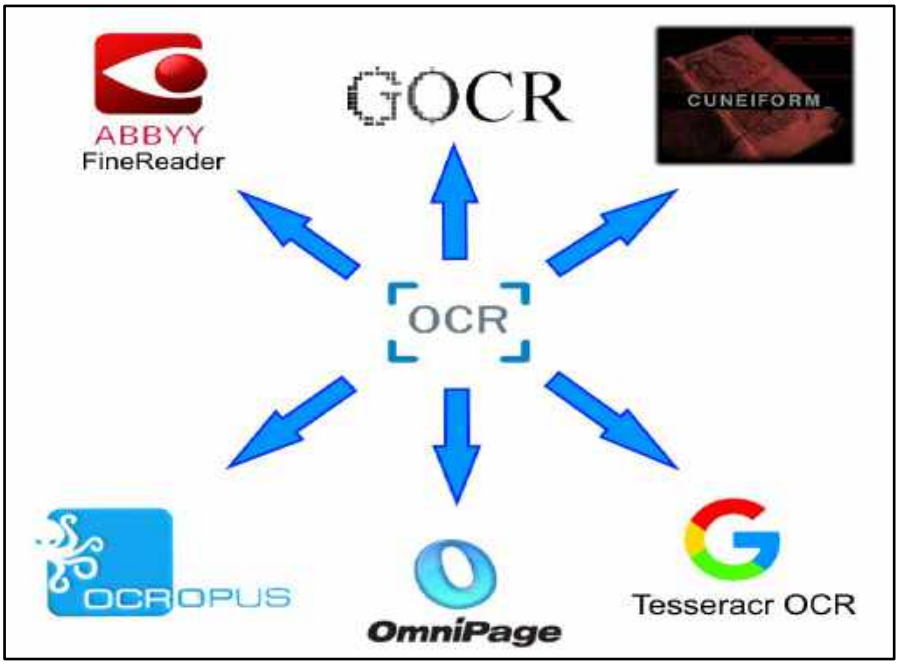


Рис. 1.2. Додатки з оптичного розпізнавання тексту

Комерційний продукт *FineReader* [13], розроблений російською компанією ABBYY − це безумовний лідер, який запатентував технологію оптичного розпізнавання символів ABBYY OCR. Її ліцензують компанії Fujitsu, Panasonic, Xerox і Samsung. Програма дає змогу перетворювати зображення друкованих документів у безліч електронних редагованих форматів. FineReader підтримує розпізнавання тексту на 190 мовах, здійснює вбудовану перевірку орфографії, має великий набір функцій для попереднього оброблення зображення. Але основна перевага цієї програми − висока точність розпізнавання. Зараз програма має більше 20 мільйонів користувачів по всьому світу.

Система оптичного розпізнавання символів *CuneiForm* російської компанії Cognitive Technologies, розроблена як комерційний продукт і анонсована 1993 року, поставлялася в комплекті з деякими моделями сканерів. Однак після кількох років перерви в розробленні в грудні 2007 року були відкриті вихідні тексти програми. Зараз CuneiForm позиціонується як незалежна від шрифту система перетворення електронних копій паперових документів і графічних файлів у редагований вигляд із можливістю збереження структури і гарнітури шрифтів оригінального документа в автоматичному чи напівавтоматичному режимі. Система містить дві програми для одиночного і пакетного оброблення електронних документів.

Система оптичного розпізнавання символів *OCRopus* спрямована на перетворення в електронний формат великого обсягу документів на базі власного ядра для розпізнавання [14]. Це програмний пакет для розпізнавання тексту, що розвивається за принципами OpenSource і поширюється під ліцензією Apache License 2.0. Програма дає змогу підключати додаткові модулі для аналізу макета вмісту, попереднього поліпшення зображення. Розробники планують за допомогою OCRopus визначати текстовий

вміст на графічних файлах по рядках і перетворювати його у звичайний текстовий формат для подальшого редагування. Планується розпізнавати і рукописні матеріали. OCRopus використовує код мови моделювання з іншого проєкту, що підтримується Google − OpenFST. Програма OCRopus наразі доступна тільки для GNU/Linux, використовує тільки інтерфейс командного рядка, приймаючи вказівки на вхідні зображення з текстом і перетворюючи дані у формат hOCR, TXT.

Рушій Tesseract − однойменна зі своїм ядром розпізнавання й вільно поширювана програма для розпізнавання текстів, що розроблялася Hewlett-Packard із середини 1980-х до середини 1990-х [15]. У 2006 році компанія Google викупила її та відкрила вихідні тексти під ліцензією Apache 2.0 для продовження роботи над перспективним проєктом. Зараз програма працює з UTF-8, підтримка мов (зокрема українська) здійснюється за допомогою установлення і настроювання додаткових модулів.

За результатами аналізу чотирьох найбільш популярних програм можна зробити такі висновки:

- низка програм не мають інтуїтивно зрозумілого інтерфейсу;
- програми не здатні працювати у фоновому режимі;
- не всі мають ресурси попереднього оброблення зображення з метою поліпшення якості;
- жодна з програм не дає 100 % результату й ідеального збереження форматування.

Усе це дає підстави для подальших досліджень щодо поліпшення результатів роботи систем розпізнавання текстів. Серед розглянутих вище засобів оптичного розпізнавання тексту як об’єкт дослідження з метою покращення загальної ефективності розпізнавання тексту було обрано програму Tesseract.

Вибір обґрунтовано тим, що програма Tesseract має відкритий вихідний код, є безкоштовною, доступною у використанні та загальні характеристики системи дають змогу отримати високу точність розпізнавання тексту, забезпечуючи відповідні умови якості зображення із текстовим документом.

Використовуючи Tesseract як інструмент, користувач може проводити окремі дослідження для виявлення основних труднощів під час розпізнавання та розроблення окремих методик їх вирішення для різноманітних варіантів вихідних зображень.

1.5. Принцип роботи та варіанти використання програми Tesseract

Tesseract можна використовувати безпосередньо або (для програмістів) за допомогою API для вилучення друкованого тексту із зображень. Програма підтримує багато мов. З версії 4.0.0 програма містить нову підсистему нейронної мережі, настроєну як розпізнавач текстових рядків. Програма походить від реалізації LSTM на основі Python OCRopus, але

була перероблена для Tesseract на C++. Мережа LSTM чудово вивчає послідовності, але дуже сповільнюється, коли надто велика кількість станів. Є емпіричні результати, які свідчать про те, що краще попросити LSTM вивчити довгу послідовність, ніж коротку послідовність багатьох класів.

Оброблення відбувається за традиційним покроковим алгоритмом, але деякі етапи є досить незвичними для подібних завдань навіть зараз.

Перший крок передбачає аналіз зв'язаних компонентів із подальшим збереженням контурів. Це було обчислювально дороге на той час дизайнерське рішення, але мало істотну перевагу: так можна було легко виявити зворотний текст і розпізнати його так само як чорно-білий текст. Tesseract був, імовірно, першим механізмом розпізнавання символів, здатним обробляти білий на чорному текст так просто. На цьому етапі контури збираються разом, шляхом вкладення, у «краплі» (blobs).

«Краплі» упорядковані в текстові рядки, а рядки і регіони аналізуються на фіксований крок або пропорційний текст. Текстові рядки поділяються на слова по-різному відповідно до типу міжсимвольного інтервалу. Текст із фіксованим кроком відразу відокремлюється символьними клітинками. Пропорційний текст поділяються на слова за допомогою означених або нечітких просторів.

Потім розпізнавання відбувається у вигляді двох проходів. У першому проході робиться спроба розпізнати кожне слово по черзі. Кожне слово, яке є задовільним, передається адаптивному класифікатору як навчальні дані. Так адаптивний класифікатор має змогу точніше розпізнавати текст внизу сторінки. У другому проході увага приділяється не усьому тексту, а окремим словам, які були визначені як погано розпізнані, після чого здійнюється повторне розпізнавання.

Останній етап виділяє нечіткі пробіли та перевіряє альтернативні гіпотези щодо висоти тексту для визначення розташування тексту з маленькими літерами.

Оскільки рушій Tesseract не має власного графічного інтерфейсу, то процес розпізнавання тексту виконується за допомогою використання командного рядка на пристрої. Так, базова команда для розпізнавання має такий вигляд:

```
tesseract imagename|stdin outputbase|stdout [options...] [configfile...],
```

де `imagename` – ім'я вхідного зображення (підтримується більшість форматів файлів зображень – усе, що читається `Leptonica`); `stdin` – інструкція для зчитування даних; `outputbase` – базове ім'я вихідного файлу (до якого буде додано відповідне розширення, за замовчуванням буде називатися `outbase.txt`); `stdout` – інструкція для відправлення вихідних даних на стандартний вихід.

Серед основних опцій (`options`) можна виділити:

- `-l lang` – мова для використання (якщо нічого не вказано, передбачено англійську мову; можливе використання одразу кількох мов при поділі мовних кодів символом «+» (наприклад, eng+ukr)), Tesseract використовує трисимвольні ISO 639-2 мовні коди;
- `--psm N` – настроювання Tesseract на виконання лише підмножини аналізу макета та приймання певної форми зображення, N може приймати значення від 0 до 10 включно;
- `--oem N` – режим роботи движка розпізнавання тексту.

Конфігурація – це відкритий текстовий файл, який містить список змінних та їх значень, по одному на рядок, з пробілом, що відділяє змінну від значення. Цікаві файли конфігурації (`configfile`) включають:

- `hocr` – результат перетворюється у формат hOCR;
- `pdf` – результат перетворюється у формат pdf замість текстового.

Також можливе використання `pytesseract – wrapper` для Tesseract-OCR Engine. У цьому випадку можно працювати із Tesseract за допомогою функціонала бібліотеки `pytesseract` на мові програмування Python.

1.6. Оцінювання й основні показники якості розпізнавання тексту

Для проведення коректних досліджень ефективності оптичного розпізнавання тексту за різних умов необхідно вибрати або сформувати кількісний показник якості. Обчислення чітко визначеного показника дасть змогу отримати первинну інформацію про ефективність оптичного розпізнавання тексту, а також проводити порівняльний аналіз для різних умов експерименту. У завданні розпізнавання тексту першочерговою є саме точність розпізнавання (Accuracy), %, для обчислення якої будемо використовувати таку формулу:

$$\mathrm{Accuracy} = \frac{\mathrm{N_{src}} - \mathrm{N_{error}} - \mathrm{N_{extra}} - \mathrm{N_{missed}}}{\mathrm{N_{src}}}\ 100\ \%, \qquad (1.1)$$

де $\mathrm{N_{src}}$ − кількість символів у початковому тексті; $\mathrm{N_{error}}$ − кількість неправильно розпізнаних символів; $\mathrm{N_{extra}}$ − кількість зайвих символів після розпізнавання; $\mathrm{N_{missed}}$ − кількість нерозпізнаних символів.

Використання цього критерію якості дає змогу враховувати не тільки кількість правильно розпізнаних символів, але і помилки під час розпізнавання, які проявляються в заміні або пропуску окремих символів і додаванні зайвих символів, яких не було в початковому тексті.

Значення $\mathrm{N_{src}}$ є строго додатним числом і залежить виключно від початкового тексту. Помилки розпізнавання $\mathrm{N_{error}}$, $\mathrm{N_{extra}}$ та $\mathrm{N_{missed}}$ є додатними числами і при цьому не мають обмежень у кількості: – $\mathrm{N_{src}} > 0$; $\mathrm{N_{error}} \geq 0$; $\mathrm{N_{extra}} \geq 0$; $\mathrm{N_{missed}} \geq 0$. Значення Accuracy обчислюється у відсотках і лежить у діапазоні від 0 % до 100 %. У випадках, коли $\mathrm{N_{error}} + \mathrm{N_{extra}} + \mathrm{N_{missed}} > \mathrm{N_{dst}}$, значення показника приймається як 0 %.

2. АЛГОРИТМИ ОБРОБЛЕННЯ ЗОБРАЖЕНЬ, ЇХ ПРОГРАМНА РЕАЛІЗАЦІЯ

У цьому розділі описано алгоритми оброблення цифрових зображень документів, які підлягають розпізнаванню. Програмно ці алгоритми реалізовані мовою програмування Python, вибір якої обумовлено наявністю великої кількості різноманітних бібліотек для оброблення зображень (серед яких бібліотека `OpenCV`) та можливістю спільного використання Python і Tesseract (рушій оптичного розпізнавання тексту з відкритим вихідним кодом) за допомогою спеціальної бібліотеки `pytesseract` [1]. Алгоритми оброблення зображень будуть подані у вигляді об'єктно-орієнтованих класів, поділених відповідно до поточного етапу оброблення. Найчастіше в описах алгоритмів наводяться окремі фрагменти коду програми з відповідними коментарями, що полегшує розуміння логіки їх роботи, також показано результати досліджень ефективності використання запропонованих алгоритмів у вигляді графіків і пояснень до них.

2.1. Основні фактори негативного впливу на роботу систем OCR

Теоретично сучасні системи оптичного розпізнавання тексту можуть досягати точності розпізнавання 98−99 %. Але важливо зазначити, що це можливо лише під час роботи з вихідними зображеннями, отриманими за ідеальних умов (наприклад, у разі сканування текстового документа високоякісним обладнанням). Досягти такої якості вихідних даних без відповідного обладнання майже неможливо, тому нас цікавлять зображення, отримані з використанням енергонезалежних пристроїв для фотографування, як-от: телефони, планшети, вебкамери та недорогі фотокамери. Отримані таким чином зображення мають низку основних недоліків, які безпосередньо впливають на точність розпізнавання тексту (рис. 2.1):

- текстовий документ займає лише певну частину всього зображення (решта – фон зображення без корисної для розпізнавання інформації);
- текстовий документ розміщений під невизначеним кутом щодо вертикальної осі зображення, оскільки майже неможливо отримати прямий кут під час фотографування;
- низька якість зображення під час фотографування, недостатня контрастність тексту;
- наявність шумів;
- нерівномірне освітлення.

Визначені недоліки впливають на кінцевий результат розпізнавання тексту. Якщо перший недолік призводить більшою мірою до уповільнення корисної роботи під час розпізнавання тексту, то другий − може стати причиною різкого зменшення точності її роботи, оскільки повернені символи вже не розпізнаються з попередньою точністю. Низька якість зображення варіюється відповідно до використаного засобу фотографування і може

взагалі не впливати на результат, тим часом як наявність шумів або нерівномірне освітлення може повністю дестабілізувати роботу системи оптичного розпізнавання тексту [16,17].

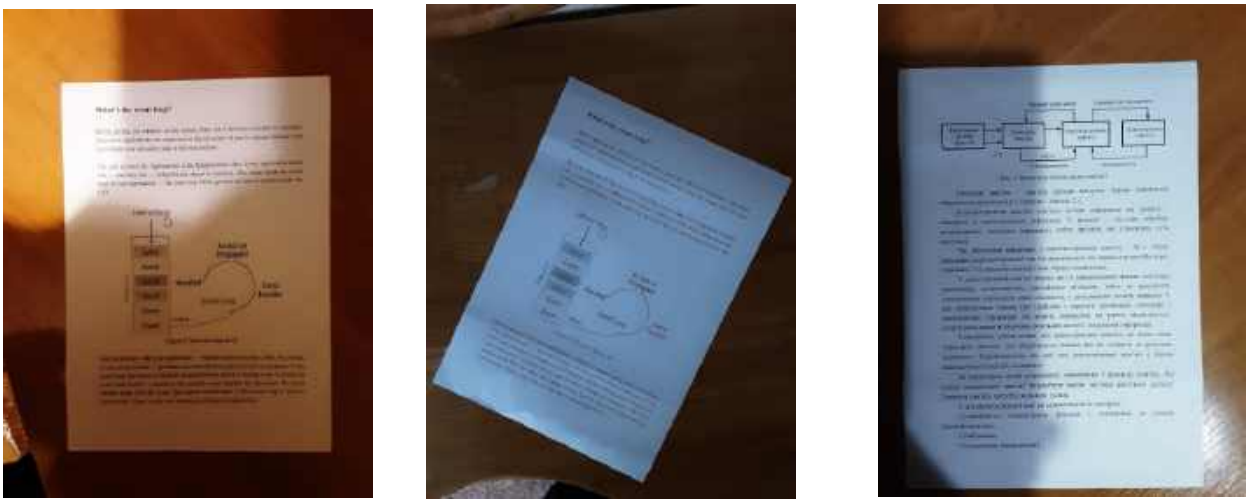


Рис. 2.1. Приклади зображень з основними недоліками

Окрім визначених вище недоліків, пов’язаних із використанням порівняно дешевих засобів фотографування, важливу роль під час розпізнавання тексту також відіграє безпосереднє наповнення текстового документа на зображенні. У цьому випадку йдеться про:

- наявність у тексті рисунків, блок-схем, таблиць тощо;
- різні шрифти та їх розмір;
- вплив курсиву, виділень, підкреслень;
- колір тексту.

У результаті можна побачити, що наявна велика кількість дестабілізаційних факторів, які так чи інакше впливають на кінцевий результат розпізнавання тексту, тому мають бути враховані під час розроблення системи оптичного розпізнавання тексту.

Для усунення визначених недоліків було вибрано три основні етапи підготовки вихідних зображень перед розпізнаванням тексту:

1) геометричні перетворення з метою виділення текстового документа на зображенні й усунення кутового відхилення;
2) попереднє оброблення тексту з метою усунення шумів, корекції контрастності тексту, освітлення сцени;
3) сегментація та анонімізація з метою фокусування на необхідних фрагментах тексту й видалення рисунків, блок-схем тощо.

Функціональну схему, що відображає сукупність і послідовність основних етапів оброблення інформації в проєктованій системі, наведено на рис. 2.2.

Далі кожен з етапів буде розглянуто більш детально з визначенням основних методів функціонування та дослідженням їх впливу на точність розпізнавання тексту.

Рис. 2.2. Функціональна схема роботи проєктованої системи

2.2. Геометричні перетворення вихідних зображень

Завдання виявлення і конвертації текстового документа в задану систему координат є ключовою в процесі роботи із зображеннями, отриманими під час фотографування текстових документів за допомогою фотокамери, телефона або планшета (зазвичай без штатива). Так, було проведено окреме дослідження впливу кута повороту текстового документа на точність розпізнавання, результат якого подано на рис. 2.3.

Можна помітити, що за умови порівняно невеликих кутів повороту (до 3°) точність розпізнавання знижується в межах 1 %, проте вже за умови 4° − до 54.66 %, а за умови 5° − до 25.46 %. Графік показує, що кут повороту текстового документа негативно впливає на ефективність роботи розпізнавача тексту.

Для вирішення завдання геометричних перетворень було розроблено алгоритм на мові Python, суть якого полягає в детектуванні контуру текстового документа на зображенні з подальшим виконанням перспективного перетворення за кутовими точками знайденого контуру.

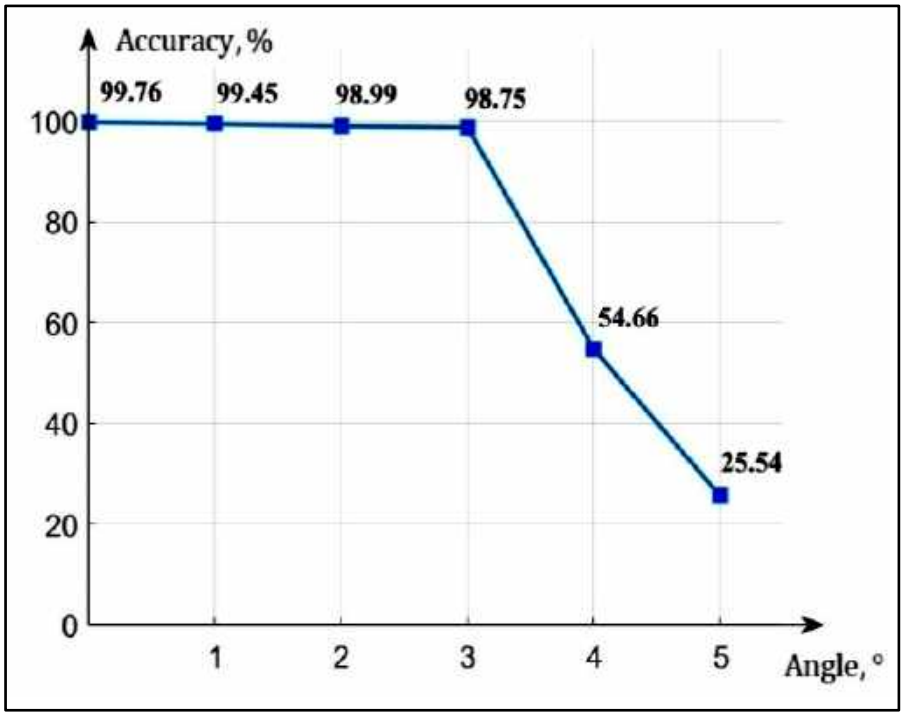


Рис. 2.3. Залежність точності розпізнавання тексту
від кута його повороту на зображенні

Під час реалізації алгоритму використані спеціальні бібліотеки OpenCV та NumPy для розширення можливостей мови Python, що додають підтримку в роботі з великими багатовимірними масивами і матрицями, а також Imutils – набір зручних функцій для полегшення основних процедур оброблення зображень (рис. 2.4).

```
1 # підключення необхідних бібліотек
2 import numpy as np
3 import cv2
4 import imutils
```

Рис. 2.4. Бібліотеки для класу GeometricTransform

Сам клас для виконання геометричних перетворень отримав назву GeometricTransform та містить чотири атрибути і шість методів. Відповідно до конструктора класу (рис. 2.5) під час створення окремих екземплярів необхідно передавати обов'язковий аргумент image – вихідне зображення текстового документа та додатковий аргумент fixed_height – фіксоване значення висоти зображення при відображенні (за замовчуванням значення – 750).

```
7  # клас для реалізації алгоритму геометричного перетворення
8  class GeometricTransform:
9
10     # конструктор класу
11     def __init__(self, image, fixed_height=750):
12         # атрибути класу
13         # фіксоване значення висоти зображення при відображенні
14         self._fixed_height = fixed_height
15         # массив для чотирьох точок (фіксований розмір зображення)
16         self._four_point_massive = []
17         # поточне зображення
18         self._original_image = image.copy()
19         # відношення висоти поточного зображення к фіксованому значенню
20         self._ratio = image.shape[0] / self._fixed_height
```

Рис. 2.5. Конструктор класу GeometricTransform

Що стосується самого алгоритму геометричних перетворень, то його можна умовно поділити на три основних етапи:

1) виділення меж зображення;
2) визначення контурів зображення;
3) виконання перспективного перетворення за точками.

На першому етапі виконується попереднє оброблення вихідного зображення для подальшого виділення меж із використанням методу Кенні (рис. 2.6).

```
90      # автоматичний режим геометричного перетворення
91      def perspective(self):
92          # Перший етап - Виділення границь зображення
93          # приведення зображення до фіксованої висоти
94          fixed_image = imutils.resize(self._original_image, height=self._fixed_height)
95          # формування зображення у відтінках сірого,
96          # використання фільтра Гауса та пошук границь методом Кенні
97          gray = cv2.cvtColor(fixed_image, cv2.COLOR_BGR2GRAY)
98          gauss = cv2.GaussianBlur(gray, (5, 5), 0)
99          edged = cv2.Canny(gauss, 50, 150)
```

Рис. 2.6. Фрагмент коду виділення меж зображення

У наведеному фрагменті коду створюється копія вихідного зображення, зведена до фіксованого значення висоти (рядок 94). Далі виконується перетворення до зображення у відтінках сірого з використанням фільтра Гаусса (рядки 97–98). Результатом виконання цього етапу є зображення edged, отримане за допомогою виділення меж методом Кенні (рядок 99). Слід зауважити, що перетворення зображення у відтінки сірого значно прискорює процес оброблення зображення, а застосування фільтра Гаусса дає змогу зменшити інтенсивність зовнішнього шуму.

На наступному (другому) етапі виконується пошук контурів зображення з подальшим виділенням найбільшого контуру прямокутної форми (рис. 2.7). Важливо зазначити, що для коректної роботи другого етапу необхідно забезпечити повну видимість текстового документа на зображенні для виявлення його контуру прямокутної форми.

```
101         # Другий етап - Визначення контурів зображення
102         # пошук контурів зображення та виділення найбільших за площею
103         cnts = cv2.findContours(edged.copy(), cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
104         cnts = imutils.grab_contours(cnts)
105         cnts = sorted(cnts, key=cv2.contourArea, reverse=True)[:5]
106         succeed_flag = False
107         # цикл по кожному з отриманих контурів
108         for c in cnts:
109             # апроксимація контура
110             peri = cv2.arcLength(c, True)
111             approx = cv2.approxPolyDP(c, 0.02 * peri, True)
112             # якщо апроксимований контур має 4 точки
113             if len(approx) == 4:
114                 # то він вважається контуром текстового документа
115                 screen_cnt = approx
116                 succeed_flag = True
117                 break
```

Рис. 2.7. Фрагмент коду визначення контурів зображення

У цьому фрагменті визначається основний контур текстового документа для його подальшого виділення з фону зображення. Для цього виконується пошук контурів на отриманому раніше зображенні з виділеними межами, а також зведення отриманого результату відповідно до використовуваної версії Python (рядки 103−104). Наступний крок − сортування отриманих контурів по площі. Це дає змогу зменшити кількість контурів і працювати тільки з найбільшими за площею контурами (рядок 105). Далі здійснюється прохід по кожному з контурів, виконується його апроксимація і визначаються точки апроксимованого контуру. Якщо кількість точок дорівнює чотирьом (прямокутник), то кутові точки цього контуру зберігаються як результат (рядки 106−117).

На третьому етапі виконується перспективне перетворення за кутовими точками виявленого контуру прямокутної форми (рис. 2.8). Цей етап є завершальним і передбачає рішення другого з поставлених завдань − усунення кутового відхилення текстового документа.

```
119         # Третій етап - Виконання перспективного перетворення
120         if succeed_flag:
121             # Отримуємо масив кутових точок контура текстового документа
122             pst = screen_cnt.reshape(4, 2)
123             # нормалізація точок з урахуванням відношення до початкового зображення
124             pst = self._points_normalization(pst * self._ratio)
125             # формування кінцевих опорних точок
126             dst, max_width, max_height = self._dst_creation(pst)
127             # визначення матриці для перспективного перетворення
128             matrix = cv2.getPerspectiveTransform(pst, dst)
129             # виконання перспективного перетворення
130             warped = cv2.warpPerspective(self._original_image, matrix, (max_width, max_height))
131             # повертаємо результат перспективного перетворення
132             return warped
133         else:
134             return self._original_image.copy()
```

Рис. 2.8. Фрагмент коду перспективного перетворення

У цьому фрагменті коду виконується перетворення кутових точок контуру текстового документа, отриманого на попередньому етапі, до формату масиву 4x2 (рядок 122). Після визначення координат опорних точок виконується перспективне перетворення за таким алгоритмом:

1. Проведення нормалізації опорних точок з урахуванням визначеного раніше відношення висот початкового і фіксованого зображень. Для цього використовується допоміжний метод `_points_normalization`, результат роботи якого записується в змінну `pst` (рядок 124).

2. Обчислення ширини `maxWidth` і висоти `maxHeight` нового зображення як максимальної відстані між опорними точками та формуються кінцеві координати `dst`, до яких будуть наводитися опорні точки в процесі перспективного перетворення. Для цього використовується допоміжний метод `_dst_creation` (рядок 126).

3. Визначення матриці `matrix` для перспективного перетворення опорних точок до кінцевих координат (рядок 128) і безпосередньо виконується перспективне перетворення (рядок 130);

4. Повернення отриманого після перспективного перетворення зображення `warped` як результат роботи методу (рядки 132–134).

Для виконання завершального етапу алгоритму були використані два додаткових методи:

- `points_normalization()`, суть якого полягає в нормалізації опорних точок для забезпечення фіксованого порядку: ліва верхня точка, права верхня точка, права нижня точка, ліва нижня точка. Відповідний фрагмент коду наведено на рис. 2.9.

```
24      # метод для нормалізації точок прямокутника
25      # у певному порядку: 1. ліва верхня точка
26      #                    2. права верхня точка
27      #                    3. права нижня точка
28      #                    4. ліва нижня точка
29      @staticmethod
30      def _points_normalization(points):
31          # створення порожнього масиву для точок
32          rect = np.zeros((4, 2), dtype="float32")
33          # визначаємо суму координат (x+y) для кожної з точок
34          s = points.sum(axis=1)
35          # ліва верхня точка має найменшу суму координат
36          # права нижня точка має найбільшу суму координат
37          rect[0] = points[np.argmin(s)]
38          rect[2] = points[np.argmax(s)]
39          # визначаємо різницю координат (x-y) для кожної з точок
40          diff = np.diff(points, axis=1)
41          # права верхня точка має найменшу різницю координат
42          # ліва нижня точка має найбільшу різницю координат
43          rect[1] = points[np.argmin(diff)]
44          rect[3] = points[np.argmax(diff)]
45          # повертаємо отримані координати
46          return rect
```

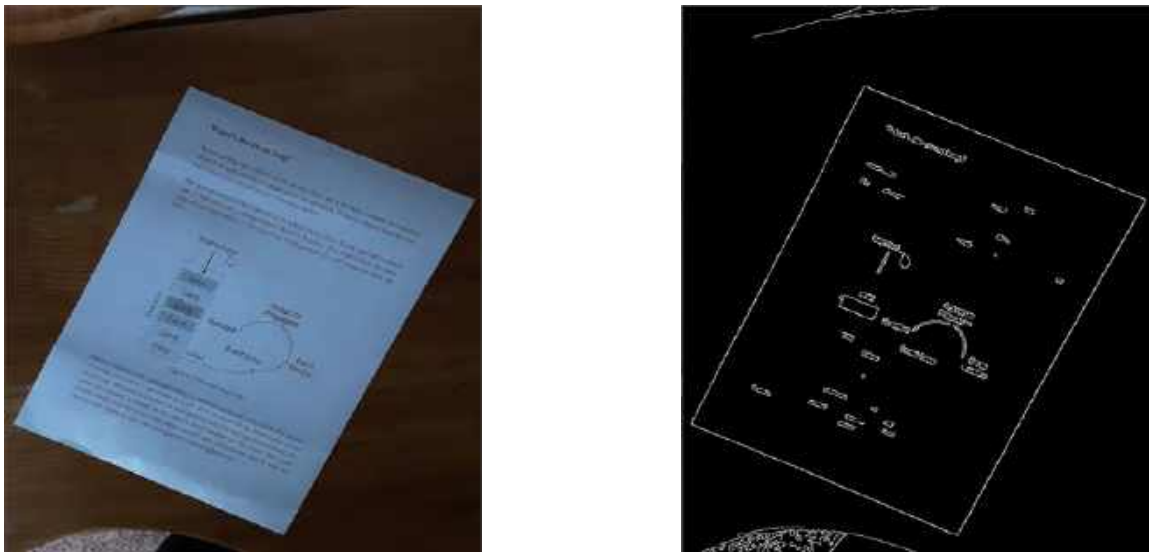
Рис. 2.9. Фрагмент коду для реалізації методу `_points_normalization()`

- `_dst_creation()`, суть якого полягає у формуванні кінцевих опорних точок для перспективного перетворення та розрахунку значень висоти й ширини нового зображення. Відповідний фрагмент коду подано на рис. 2.10.

```
48      # метод для формування кінцевих опорних точок
49      @staticmethod
50      def _dst_creation(normalized_points):
51          (tl, tr, br, bl) = normalized_points
52          # розрахунок ширини нового зображення, як максимальну
53          # відстань між x-координатами крайніх  точок прямокутника
54          width_a = np.sqrt(((br[0] - bl[0]) ** 2) + ((br[1] - bl[1]) ** 2))
55          width_b = np.sqrt(((tr[0] - tl[0]) ** 2) + ((tr[1] - tl[1]) ** 2))
56          max_width = max(int(width_a), int(width_b))
57          # розрахунок висоти нового зображення, як максимальну
58          # відстань між y-координатами крайніх точок прямокутника
59          height_a = np.sqrt(((tr[0] - br[0]) ** 2) + ((tr[1] - br[1]) ** 2))
60          height_b = np.sqrt(((tl[0] - bl[0]) ** 2) + ((tl[1] - bl[1]) ** 2))
61          max_height = max(int(height_a), int(height_b))
62          # Формування кінцевих координат опорних точок
63          dst = np.array([
64                  [0, 0],  # ліва верхня точка
65                  [max_width - 1, 0],  # права верхня точка
66                  [max_width - 1, max_height - 1],  # права нижня точка
67                  [0, max_height - 1]], dtype="float32")  # ліва нижня точка
68          return dst, max_width, max_height
```

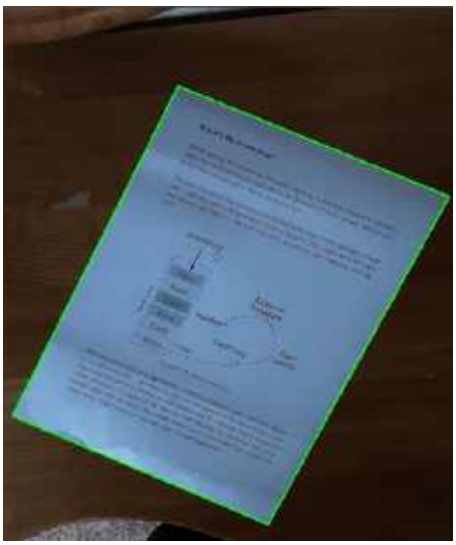
Рис. 2.10. Фрагмент коду для реалізації методу `_dst_creation()`

На рис. 2.11 показано результати роботи алгоритму на кожному з етапів роботи для конкретного зображення.

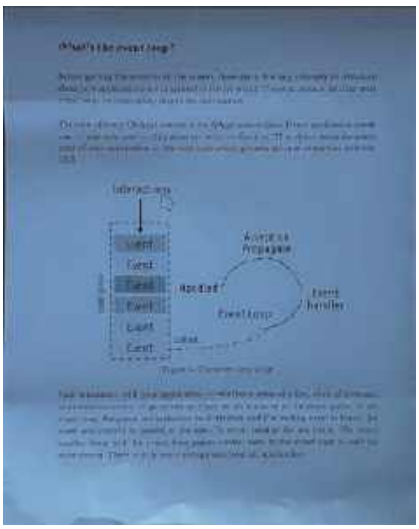


а *б*

Рис. 2.11. Візуалізація поетапної роботи алгоритму:
а − вихідне зображення; *б* − визначення меж;
в − виділення контуру; *г* – перспективне перетворення



в



г

Рис. 2.11. Закінчення

Ефективність роботи цього алгоритму було підтверджено шляхом проведення дослідження на розглянутих раніше зображеннях, повернутих на кути від 1° до 5°. Результати дослідження наведені на рис. 2.12. Використання алгоритму геометричних перетворень дає змогу досягти високого значення точності розпізнавання та її стійкості незалежно від початкового кута відхилення текстового документа на зображенні. Алгоритм дає змогу не тільки підтримувати постійне значення точності розпізнавання за умови різних кутів відхилення, але й прибирає непотрібний фон на зображенні (рис. 2.11, *г*). У нашому випадку в процесі геометричних перетвореннь похибка не перевищує 2°. Цього достатньо для підтримання високої точності розпізнавання текстів.

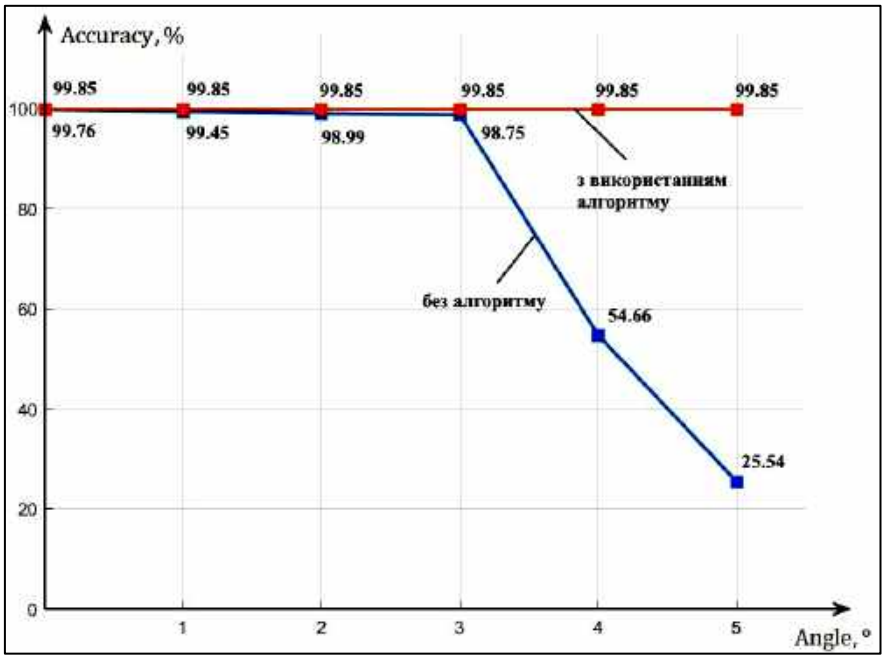
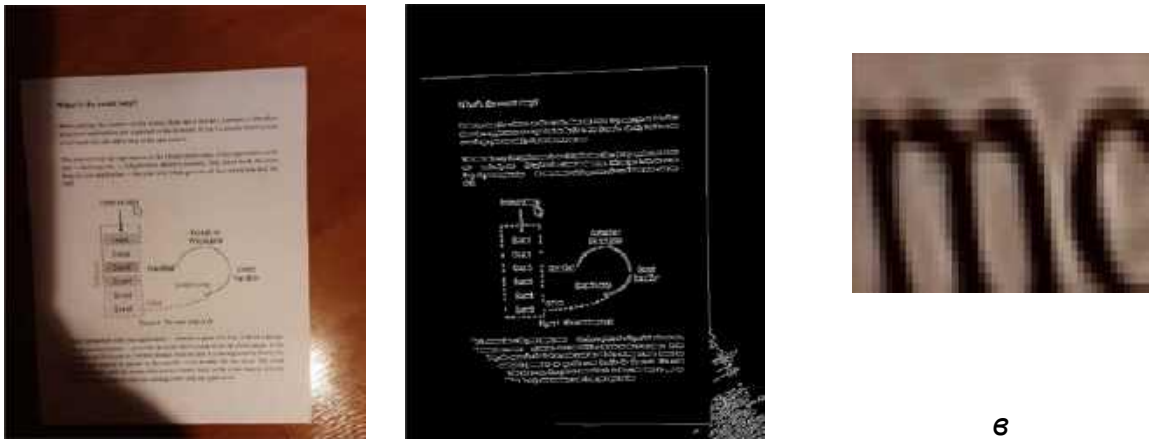


Рис. 2.12. Графіки залежності точності розпізнавання тексту від кута повороту документа

Слід зазначити, що не всі зображення однаково добре підходять для застосування алгоритму. Наприклад, якщо один із фрагментів зображення занадто яскравий або навпаки затінений, то алгоритм не може обвести потрібний прямокутний контур; виникають специфічні похибки геометричного перетворення (рис. 2.13).



а *б* *в*

Рис. 2.13. Геометрична трансформація за умови нерівномірного освітлення зображення: *а* − вихідне зображення; *б* − результат виділення меж; *в* − результат перспективного перетворення

Добре видно, що порушення контурів зображення (рис. 2.13, *б*) призвело до некоректної ідентифікації головного прямокутного контуру (рис. 2.13, *в*). Оскільки етап геометричного перетворення є базовим у загальному алгоритмі оброблення зображень для розпізнавання тексту, помилка на цьому етапі є неприпустимою.

Для розв'язання цієї проблеми було реалізовано можливість додаткових інтерактивних геометричних перетворень. Основне завдання інтерактивної процедури – самостійне виділення на зображенні чотирьох кутових точок текстового документа, на основі яких у подальшому буде проводитися перспективне перетворення зображення. Метод, необхідний для реалізації інтерактивного режиму геометричних перетворень, показано на рис. 2.14.

У цьому методі створюється спеціальне вікно з відображенням поточного зображення та можливістю задати чотири кутові точки текстового документа за допомогою комп'ютерної миші (рядки 139−143). Крім того, в інтерактивному режимі є можливість скасувати останню операцію (натиснувши «b») і повністю очистити масив точок (натиснувши «r») (рядки 146−155). Натискання «enter» перевіряє кількість вибраних точок у відповідному масиві (потрібно чотири точки), масштабує їх відповідно до вихідних розмірів зображення і виконує всі необхідні кроки для виконання перспективного перетворення (рядки 157−172). Після цього інтерактивне вікно закривається, а змінна warped повертається як результат роботи (рядки 174−180).

```
136     # метод для інтерактивного режиму геометричного перетворення
137     def interactive_perspective(self):
138         # створюємо іменоване вікно для зображення
139         cv2.namedWindow("PerspectiveWindow")
140         # закріплюємо за іменованим вікном обробник
141         cv2.setMouseCallback("PerspectiveWindow", self._point_detection)
142         # відображення зображення
143         cv2.imshow('PerspectiveWindow', imutils.resize(self._original_image, height=self._fixed_height))
144         # робочий цикл
145         while True:
146             key = cv2.waitKey(1)
147             # при натисканні клавіши 'r' повертаємо початквое зображення
148             if key == ord("r"):
149                 self._four_point_massive.clear()  # очищуємо масив точок
150                 self._point_display()  # викликаємо функцію для відображення
151             # при натисканні  клавіши 'b' видаляємо останню виконану операцію
152             elif key == ord("b"):
153                 if len(self._four_point_massive) > 0:  # якщо є хоч одна точка в масиві
154                     self._four_point_massive.pop()  # видаляємо останню точку
155                     self._point_display()  # викликаємо функцію для відображення
156             # при натисканні  клавіши 'enter' закриваємо робоче вікно
157             elif key == 13:
158                 if len(self._four_point_massive) == 4:
159                     pst = np.zeros((4, 2), dtype="float32")
160                     i = 0
161                     # масштабування точок в масивах під початкове зображення
162                     for point in self._four_point_massive:
163                         pst[i] = point
164                         i += 1
165                     # нормалізація точок з урахуванням відношення до початкового зображення
166                     pst = self._points_normalization(pst * self._ratio)
167                     # формування кінцевих опорних точок
168                     dst, max_width, max_height = self._dst_creation(pst)
169                     # визначення матриці для перспективного перетворення
170                     matrix = cv2.getPerspectiveTransform(pst, dst)
171                     # виконання перспективного перетворення
172                     warped = cv2.warpPerspective(self._original_image, matrix, (max_width, max_height))
173                     # закриваємо вікно
174                     cv2.destroyAllWindows()
175                     # повертаємо результат перспективного перетворення
176                     return warped
177                 else:
178                     # закриваємо вікно
179                     cv2.destroyAllWindows()
180                     return self._original_image.copy()
```

Рис. 2.14. Фрагмент коду для реалізації методу `interactive_perspective()`

Окрім розглянутих раніше допоміжних методів `_points_normalization()` та `_dst_creation()`, використовуються також такі:

1. `_point_display()` – для відображення точок на зображенні, вибраних користувачем (рис. 2.15).

2. `_point_detection()` – для оброблення подій (після натискання лівої кнопки миші) та фіксування поточних координат точки, а також для виклику `point_display()` для відображення точки (рис. 2.16).

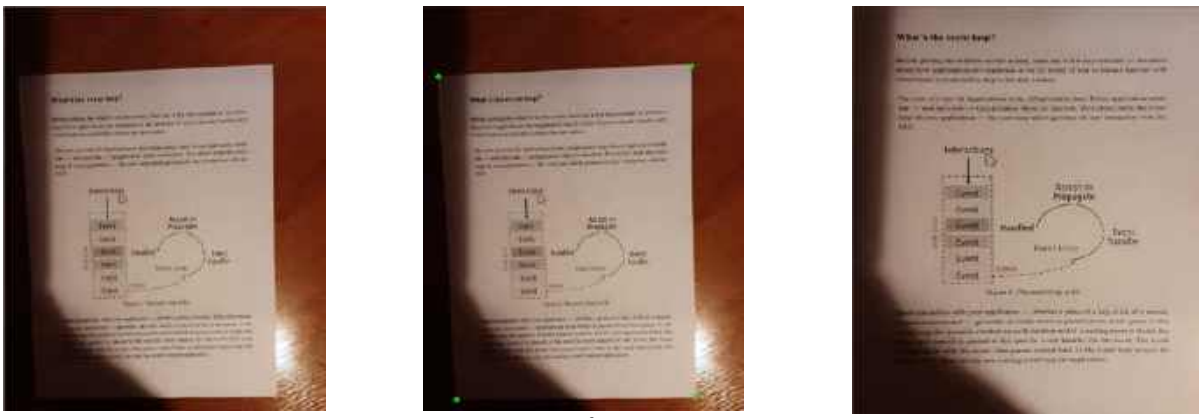
У результаті користувач отримує можливість виконувати етап геометричних перетворень в інтерактивному режимі з ручним позначенням кутових точок на зображенні текстового документа. Результат роботи інтерактивного режиму наведено на рис. 2.17.

```
70      # метод для відрисовки точок у інтерактивному режимі
71      def _point_display(self):
72          # отримуємо копію поточного зображення
73          tmp_image = imutils.resize(self._original_image, height=self._fixed_height)
74          # для кожного набору точок масиву
75          for point in self._four_point_massive:
76              # малюємо прямокутник одиничної товщини
77              cv2.circle(tmp_image, point, 3, (0, 255, 0), 2)
78          # оновлюємо головне вікно для демонстрації
79          cv2.imshow("PerspectiveWindow", tmp_image)
```

Рис. 2.15. Фрагмент коду для реалізації методу `_point_display()`

```
81      # метод для обробки подій натискання миші в інтерактивному режимі
82      def _point_detection(self, event, x, y, flags, params):
83          # при натисканні лівої кнопки миші
84          if event == cv2.EVENT_LBUTTONDOWN and len(self._four_point_massive) < 4:
85              tmp_point = (int(x), int(y))  # запам'ятовуємо координати точки
86              self._four_point_massive.append(tmp_point)  # додаємо точку у масив
87          # викликаємо функцію для відмальовування геометричних примітивів
88          self._point_display()
```

Рис. 2.16. Фрагмент коду для реалізації методу `_point_detection()`



а *б* *в*

Рис. 2.17. Результати інтерактивних геометричних перетворень: *а* – вихідне зображення; *б* − ідентифікація кутових точок текстового документа; *в* − результат геометричних перетворень

Таким чином, індивідуальне виконання етапу геометричних перетворень залежить від сталості освітлення в оригінальному документі. Якщо зображення рівномірно освітлене, то можна використовувати автоматичний алгоритм, а у випадках із нерівномірним освітленням і затемненням певних сегментів − інтерактивний режим.

2.3. Попереднє оброблення зображень

Існує низка факторів, що негативно впливають на якість зображення і, як наслідок, на точність розпізнавання тексту. Це різні шуми, погані умови освітлення під час фотографування документів, недостатня контрастність знімків, помилки оператора в процесі установлення фокусної відстані тощо.

Щоб оцінити вплив цих факторів, проведено чимало досліджень, які показали ступінь їх негативного впливу. Наприклад, оцінювався вплив імпульсного шуму на якість розпізнавання тексту. Результати аналізу подано на рис. 2.18.

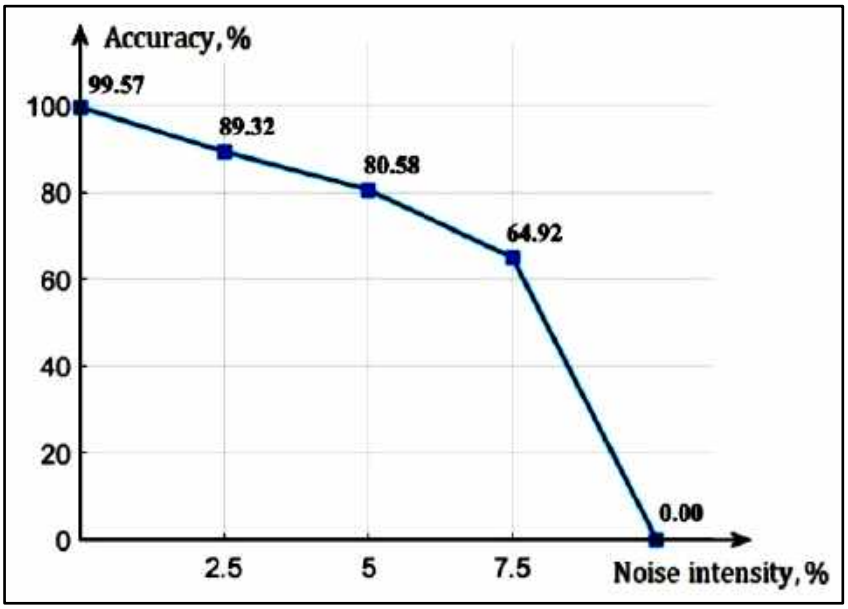


Рис. 2.18. Залежність точності розпізнавання тексту від інтенсивності імпульсного шуму

Очевидно, що навіть невелика інтенсивність шуму на зображенні (2,5 %) знижує точність розпізнавання на 10 %, а якщо інтенсивність понад 10 %, то розпізнавач взагалі перестає працювати.

Щоб усунути вплив шуму й інших зовнішніх чинників що перешкоджають, використовуються ресурси бібліотеки OpenCV, яка містить різноманітний набір функцій для оброблення зображень. Структуру загального алгоритму, покликаного забезпечити максимальну точність розпізнавання за умови дії дестабілізаційних впливів, наведено на рис. 2.19.

Важливо зазначити, що показані операції працюють із зображеннями у відтінках сірого. Тому перед застосуванням алгоритму необхідно перетворити колірний простір вихідного зображення.

Для реалізації алгоритму розроблений спеціальний клас PreProcessing, у якому наявні всі методи, необхідні для роботи алгоритму. Для функціонування класу використовуються бібліотеки OpenCV, NumPy та Matplotlib (рис. 2.20).

Рис. 2.19. Алгоритм попереднього оброблення зображення

```
1 # підключення необхідних бібліотек
2 import numpy as np
3 import cv2
4 from matplotlib import pyplot as plt
```

Рис. 2.20. Бібліотеки для класу PreProcessing

Усі дослідження роботи окремих методів попереднього оброблення проводилися на зображенні з імпульсним шумом інтенсивністю 5 %.

2.3.1. Перетворення колірного простору

Під час зчитування зображень за допомогою функцій бібліотеки OpenCV вихідне зображення має формат BGR (синій-зелений-червоний). Це зображення необхідно перетворити у відтінки сірого для можливості виконання подальших операцій. Також виконання перетворення обмежує кількість робочих каналів з трьох до одного, що значно скорочує час оброблення.

Для перетворення колірного простору зображення використовується спеціальна функція бібліотеки OpenCV, яка має такий синтаксис:

```
cv2.cvtColor(src,code[,dst[,dstCn]]),
```

де src – початкове зображення, колірний простір якого потрібно змінити; code – код перетворення колірного простору; dst – вихідне зображення того самого розміру і глибини, що і зображення src (це необов'язковий параметр); dstCn – кількість каналів у кінцевому зображенні, якщо параметр дорівнює 0, то кількість каналів визначається автоматично з src на code (це необов'язковий параметр).

Для перетворення зображення BGR-формату на зображення у відтінках сірого (grayscale) використовується значення cv2.COLOR_BGR2GRAY для параметра code.

2.3.2. Фільтрація шумів

На якість зображень впливають різні шуми в процесі передавання їх каналами зв'язку, а також на етапі формування. Тому першим етапом попереднього оброблення зображень є фільтрація. Існує велика кількість різних типів фільтрації, кожен із яких має свою сферу застосування. Що стосується роботи із зображеннями, то були вибрані медіанний фільтр, розмивання Гаусса та білатеральний фільтр.

Медіанний фільтр − це один із видів цифрових фільтрів, широко використовуваний у цифровму обробленні сигналів і зображень для зменшення рівня шуму. Значення відліків у середині вікна фільтра сортуються в порядку збільшення (зменшення); і значення в середині упорядкованого списку надходить на вихід фільтра. У разі парного числа відліків у вікні вихідне значення фільтра дорівнює середньому значенню двох відліків у середині впорядкованого списку. Вікно переміщається вздовж сигналу, що фільтрується, і обчислення повторюються. Медіанна фільтрація − ефективна процедура оброблення сигналів, що зазнають впливу імпульсних перешкод.

Розмивання Гаусса – це тип фільтра розмивання зображення, що використовує функцію Гаусса (яка також трапляється в нормальному статистичному розподілі) для розрахунку трансформації кожного пікселя в зображенні. Коли метод застосовується у двох вимірах, отримується поверхня, контури якої є концентричними колами розподілу Гаусса із центральної точки. Значення із цього розподілу використовуються для створення матриці згортки. Для кожного нового значення пікселя визначається середнє зважене в околі пікселя. Значення поточного оригінального пікселя має більшу вагу (найвище значення розподілу Гаусса), а сусідні пікселі отримують все меншу вагу залежно від того, наскільки далеко вони розміщені від поточного оригінального пікселя. Це надає ефект розмитості, яка зберігає межі краще, ніж інші аналогічні фільтри розмиття.

Білатеральний фільтр − це нелінійний фільтр, у якому вага кожного пікселя обчислюється шляхом множення ядра фільтра нижніх частот у просторовій області на функцію впливу в яскравісній області, яка зменшує

вагу пікселя з великою різницею за яскравістю. Цей фільтр застосовує просторове зважене осереднення без згладжування країв. Це досягається шляхом комбінування двох гауссівських фільтрів: один фільтр працює в просторовій області, а інший − в області інтенсивності (яскравості). Таким чином, не лише просторова відстань, а й відстань інтенсивності також важлива для визначення ваг. Фільтр широко використовується для заглушення шумів зі збереженням контурів.

При реалізації класу було прийнято, що поточний тип фільтра може бути змінений користувачем у процесі для максимізації гнучкості системи під час роботи з різними зображеннями. Реалізацію методу фільтрації шумів наведено на рис. 2.21.

У самому методі виконується спочатку перевірка необхідності фільтрації шумів через логічну змінну `is_required` з подальшим аналізом поточного типу фільтра, заданого користувачем через змінну `_f_type`. Після визначення поточного типу фільтра викликається спеціальна функція з бібліотеки `OpenCV` із відповідними параметрами.

```
51    # фільтрація шумів
52    def _noise_filtration(self, is_required):
53        if is_required:  # якщо операція необхідна
54            # Перевірка методу фільтрації
55            if self._f_type == 'Median':  # медіанний фільтр
56                self._filtration_image = cv2.medianBlur(self._grayscale_image, self._m_ksize)
57            elif self._f_type == 'Gaussian':  # фільтр Гауса
58                self._filtration_image = cv2.GaussianBlur(self._grayscale_image, (self._g_ksize, self._g_ksize),
59                                                          self._g_sigmax, self._g_sigmay)
60            elif self._f_type == 'Bilateral':  # двосторонній фільтр
61                self._filtration_image = cv2.bilateralFilter(self._grayscale_image, self._b_d, self._b_sigma_color,
62                                                             self._b_sigma_space)
63            else:  # при непрвильних параметрах налаштування
64                self._filtration_image = self._grayscale_image.copy()
65        else:
66            self._filtration_image = self._grayscale_image.copy()
```

Рис. 2.21. Фрагмент коду для реалізації методу фільтрації шумів

Кожен із поданих фільтрів було протестовано на ефективність зменшення рівня імпульсного шуму. Результати дослідження наведено на рис. 2.22.

Отже, медіанний фільтр найбільш ефективний щодо зменшення рівня імпульсного шуму. Використуючи такий фільтр, точність розпізнавання становить 98,84 %, тоді як для вихідного зображення у відтінках сірого − 89,32 %.

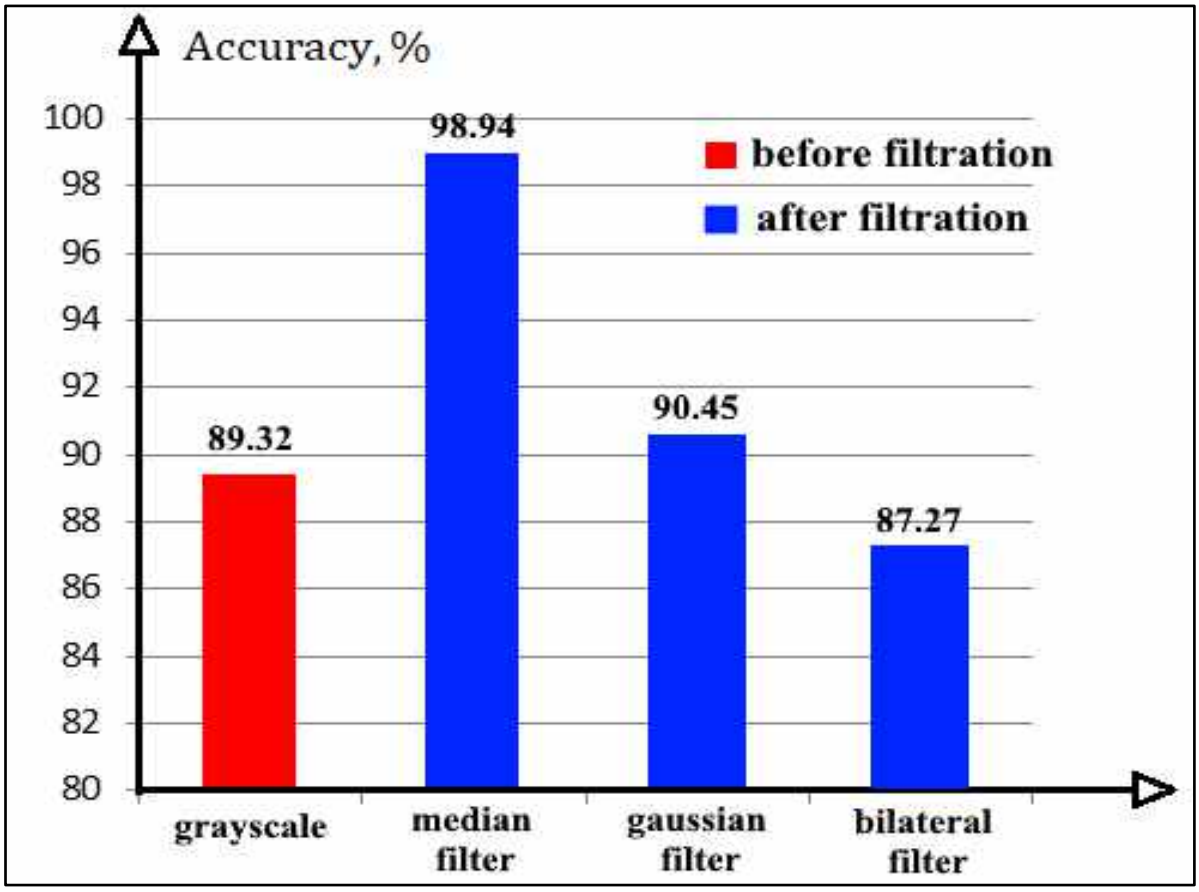


Рис. 2.22. Порівняльна діаграма ефективності різних фільтрів

2.3.3. Підвищення різкості зображення

Підвищення різкості зображення може бути досягнуто в частотній області за допомогою процедури високочастотної фільтрації, яка пригнічує низькочастотні складові. Отримані в результаті високочастотної фільтрації зображення мають одну загальну властивість: середнє значення яскравості фону наближається до нуля. Це виникає внаслідок того, що такі фільтри знищують нульову компоненту їх перетворення Фур'є.

Використання високочастотних фільтрів у попередньому обробленні зображень дає змогу підкреслити межі окремих текстових символів, які були розмиті внаслідок фільтрації шумів на попередньому етапі оброблення. При цьому використовуються такі типові маски:

$$\mathrm{H}_1 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}, \quad \mathrm{H}_2 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}, \quad \text{або } \mathrm{H}_3 = \begin{bmatrix} 1 & -2 & 1 \\ -2 & 5 & -2 \\ 1 & -2 & 1 \end{bmatrix}.$$

На рис. 2.23 зображено реалізацію методу високочастотної фільтрації для підвищення різкості зображень. Цей метод працює аналогічно методу фільтрації шумів: після перевірки необхідності виконання операції за допомогою логічної змінної `is_required` виконується перевірка змінної `_hf_type` для визначення та створення поточного типу маски для фільтрації.

```
68      # високочастотна фільтрація
69      def _hf_filtration(self, is_required):
70          if is_required:  # якщо операція необхідна
71              # Перевірка методу високочастотної фільтрації
72              if self._hf_type == "H1":
73                  mask = np.array([[-1, -1, -1], [-1, 9, -1], [-1, -1, -1]])
74              elif self._hf_type == "H2":
75                  mask = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
76              elif self._hf_type == "H3":
77                  mask = np.array([[1, -2, 1], [-2, 5, -2], [1, -2, 1]])
78              else:  # при непрвильних параметрах налаштування
79                  mask = np.array([[1, 1, 1], [1, 1, 1], [1, 1, 1]])
80              # використання функції OpenCV для фільтрації за маскою
81              self._hf_filtration_image = cv2.filter2D(self._filtration_image, -1, mask)
82          else:
83              self._hf_filtration_image = self._filtration_image
```

Рис. 2.23. Фрагмент коду для реалізації методу високочастотної фільтрації

Кожен із доступних фільтрів перевірявся на ефективність за умови підвищення різкості зображення. Результати дослідження подано на рис. 2.24.

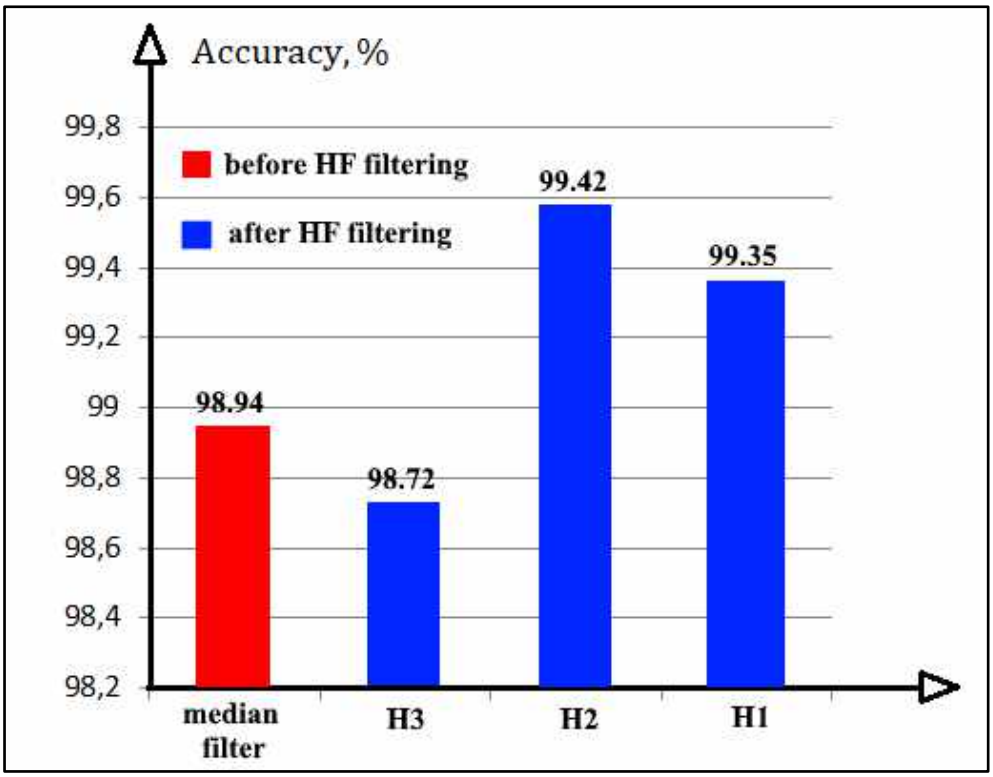


Рис. 2.24. Порівняльна діаграма ефективності різних високочастотних фільтрів

2.3.4. Бінаризація зображень

Бінаризація дає змогу привести текстові символи на зображенні до єдиного рівня чорного кольору, збільшуючи при цьому контраст друкованого тексту на білому тлі. Найважливішим аспектом у забезпеченні якості бінарізації є встановлення порога. Якщо яскравість пікселя більша за поро-

гове значення, йому присвоюється перше з двох значень (білий), в іншому випадку – інше значення (чорний). У бібліотеці OpenCV для цього використовується функція cv2.threshold. Першим її аргументом є вихідне зображення сірого кольору, другим аргументом – порогове значення, яке використовується для класифікації яскравості пікселів. Третій параметр maxVal являє собою максимальне значення яскравості пікселя (зазвичай становить 255), яке присвоюється відповідно до рішення поточного завдання. OpenCV також надає різні варіанти визначення порогових значень, що присвоюються четвертим параметром функції:

- cv2.THRESH_BINARY;
- cv2.THRESH_BINARY_INV;
- cv2.THRESH_TRUNC;
- cv2.THRESH_TOZERO;
- cv2.THRESH_TOZERO_INV.

Для бімодальних зображень, гістограма яких має два різних піки, як порогове значення може набути значення десь між цими піками. Цей підхід використовується в процесі бінаризації Otsu. Порогове значення для бімодальних зображень автоматично розраховується за допомогою гістограми зображення. Якщо зображення не є бімодальним, то бінаризація не буде точною.

Бінаризація Otsu використовує функцію cv2.threshold(), отримуючи при цьому додатковий прапор cv2.THRESH_OTSU. Порогове значення спочатку встановлюється таким, що дорівнює нулю. Потім алгоритм автоматично визначає оптимальний поріг і повертає retVal як другий вихідний параметр. Якщо порогове значення Otsu не використовується, retVal відповідає пороговому значенню, що використовувалося раніше.

Використання фіксованих порогів не завжди є продуктивним. Наприклад, якщо області зображення мають дуже різну освітленість, то це призводить до небажаних результатів бінаризації. У таких випадках використовуються адаптивні методи бінаризації, де алгоритм обчислює окремі значення порогів для невеликих областей зображення. У цих випадках можна використовувати такі функції OpenCV:

- cv2.ADAPTIVE_THRESH_MEAN_C: – установлення порога на основі середніх значень яскравості суміжних областей зображення;
- cv2.ADAPTIVE_THRESH_GAUSSIAN_C: – порогове значення, що є зваженою сумою значень суміжних областей, де ваги є вікном Гаусса, тут розмір блока визначає розмір області, а C – константа, що обчислюється з попередньо розрахованого середнього або зваженого середнього.

На рис. 2.25 наведено реалізацію методу бінаризації, а на рис. 2.26 показано результати дослідження ефективності для кожного типу бінаризації; засвічення зображень вважалося нормальним.

```
86      def _binarization(self, is_required):
87          if is_required:  # якщо операція необхідна
88              # Перевірка методу бінаризації
89              # порогова бінаризація методом THRESH_BINARY
90              if self._b_type == "Simple" and self._s_type == "THRESH_BINARY":
91                  self._binarization_image = cv2.threshold(self._hf_filtration_image, self._s_thresh, self._s_max, cv2.THRESH_BINARY)[1]
92              # порогова бінаризація методом THRESH_BINARY_INV
93              elif self._b_type == "Simple" and self._s_type == "THRESH_BINARY_INV":
94                  self._binarization_image = cv2.threshold(self._hf_filtration_image, self._s_thresh, self._s_max, cv2.THRESH_BINARY_INV)[1]
95              # порогова бінаризація методом THRESH_TRUNC
96              elif self._b_type == "Simple" and self._s_type == "THRESH_TRUNC":
97                  self._binarization_image = cv2.threshold(self._hf_filtration_image, self._s_thresh, self._s_max, cv2.THRESH_TRUNC)[1]
98              # порогова бінаризація методом THRESH_TOZERO
99              elif self._b_type == "Simple" and self._s_type == "THRESH_TOZERO":
100                 self._binarization_image = cv2.threshold(self._hf_filtration_image, self._s_thresh, self._s_max, cv2.THRESH_TOZERO)[1]
101             # порогова бінаризація методом THRESH_TOZERO_INV
102             elif self._b_type == "Simple" and self._s_type == "THRESH_TOZERO_INV":
103                 self._binarization_image = cv2.threshold(self._hf_filtration_image, self._s_thresh, self._s_max, cv2.THRESH_TOZERO_INV)[1]
104             # бінаризація Отсу методом THRESH_BINARY
105             elif self._b_type == "Otsu" and self._O_type == "THRESH_BINARY":
106                 self._binarization_image = cv2.threshold(self._hf_filtration_image, 0, self._o_max, cv2.THRESH_BINARY + cv2.THRESH_OTSU)[1]
107             # бінаризація Отсу методом THRESH_BINARY_INV
108             elif self._b_type == "Otsu" and self._O_type == "THRESH_BINARY_INV":
109                 self._binarization_image = cv2.threshold(self._hf_filtration_image, 0, self._o_max, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)[1]
110             # бінаризація адаптивним методом (mean) з THRESH_BINARY
111             elif self._b_type == "Adaptive" and self._a_type1 == "ADAPTIVE_MEAN_C" and self._a_type2 == "THRESH_BINARY":
112                 self._binarization_image = cv2.adaptiveThreshold(self._hf_filtration_image, self._a_max, cv2.ADAPTIVE_THRESH_MEAN_C, \
113                                                                  cv2.THRESH_BINARY, self._a_ksize, self._a_c)
114             # бінаризація адаптивним методом (mean) з THRESH_BINARY_INV
115             elif self._b_type == "Adaptive" and self._a_type1 == "ADAPTIVE_MEAN_C" and self._a_type2 == "THRESH_BINARY_INV":
116                 self._binarization_image = cv2.adaptiveThreshold(self._hf_filtration_image, self._a_max, cv2.ADAPTIVE_THRESH_MEAN_C, \
117                                                                  cv2.THRESH_BINARY_INV, self._a_ksize, self._a_c)
118             # бінаризація адаптивним методом (gaussian) з THRESH_BINARY
119             elif self._b_type == "Adaptive" and self._a_type1 == "ADAPTIVE_GAUSSIAN_C" and self._a_type2 == "THRESH_BINARY":
120                 self._binarization_image = cv2.adaptiveThreshold(self._hf_filtration_image, self._a_max, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, \
121                                                                  cv2.THRESH_BINARY, self._a_ksize, self._a_c)
122             # бінаризація адаптивним методом (gaussian) з THRESH_BINARY_INV
123             elif self._b_type == "Adaptive" and self._a_type1 == "ADAPTIVE_GAUSSIAN_C" and self._a_type2 == "THRESH_BINARY_INV":
124                 self._binarization_image = cv2.adaptiveThreshold(self._hf_filtration_image, self._a_max, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, \
125                                                                  cv2.THRESH_BINARY_INV, self._a_ksize, self._a_c)
126             else:  # при непрвильних параметрах налаштування
127                 self._binarization_image = self._hf_filtration_image
128         else:
129             self._binarization_image = self._hf_filtration_image
```

Рис. 2.25. Фрагмент коду для реалізації методу бінаризації

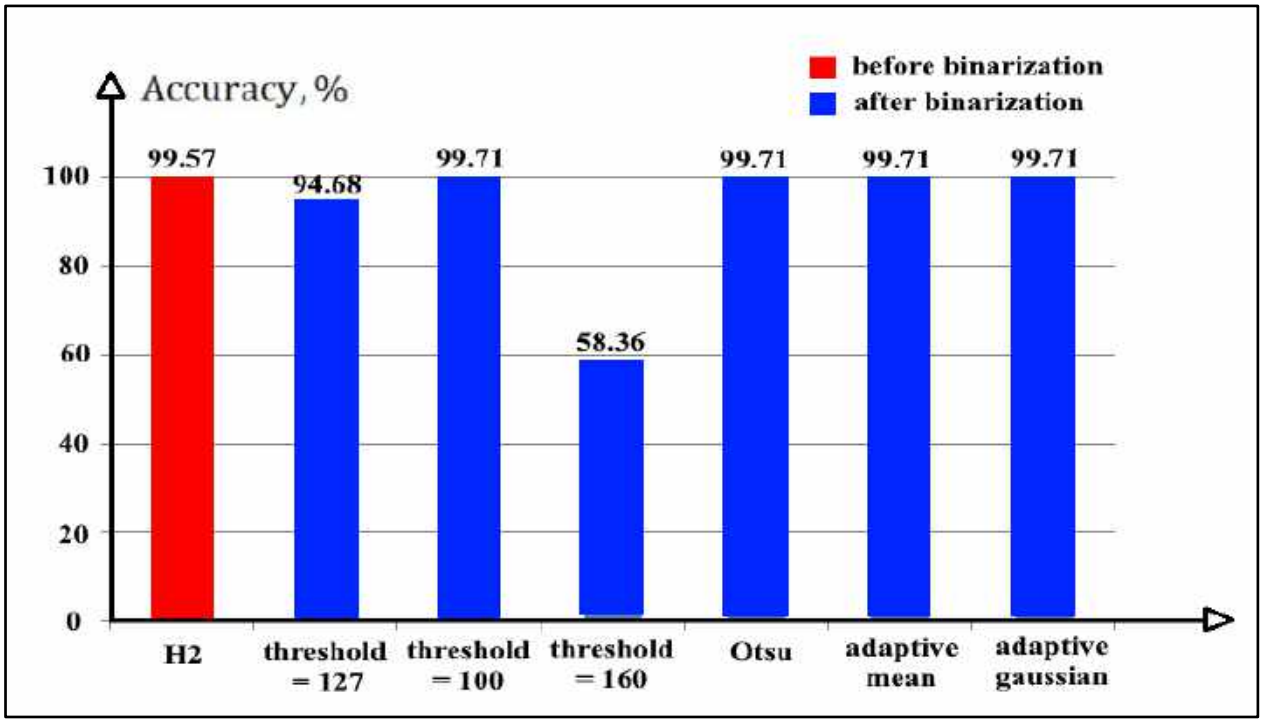


Рис. 2.26. Порівняльна характеристика ефективності типів бінаризації

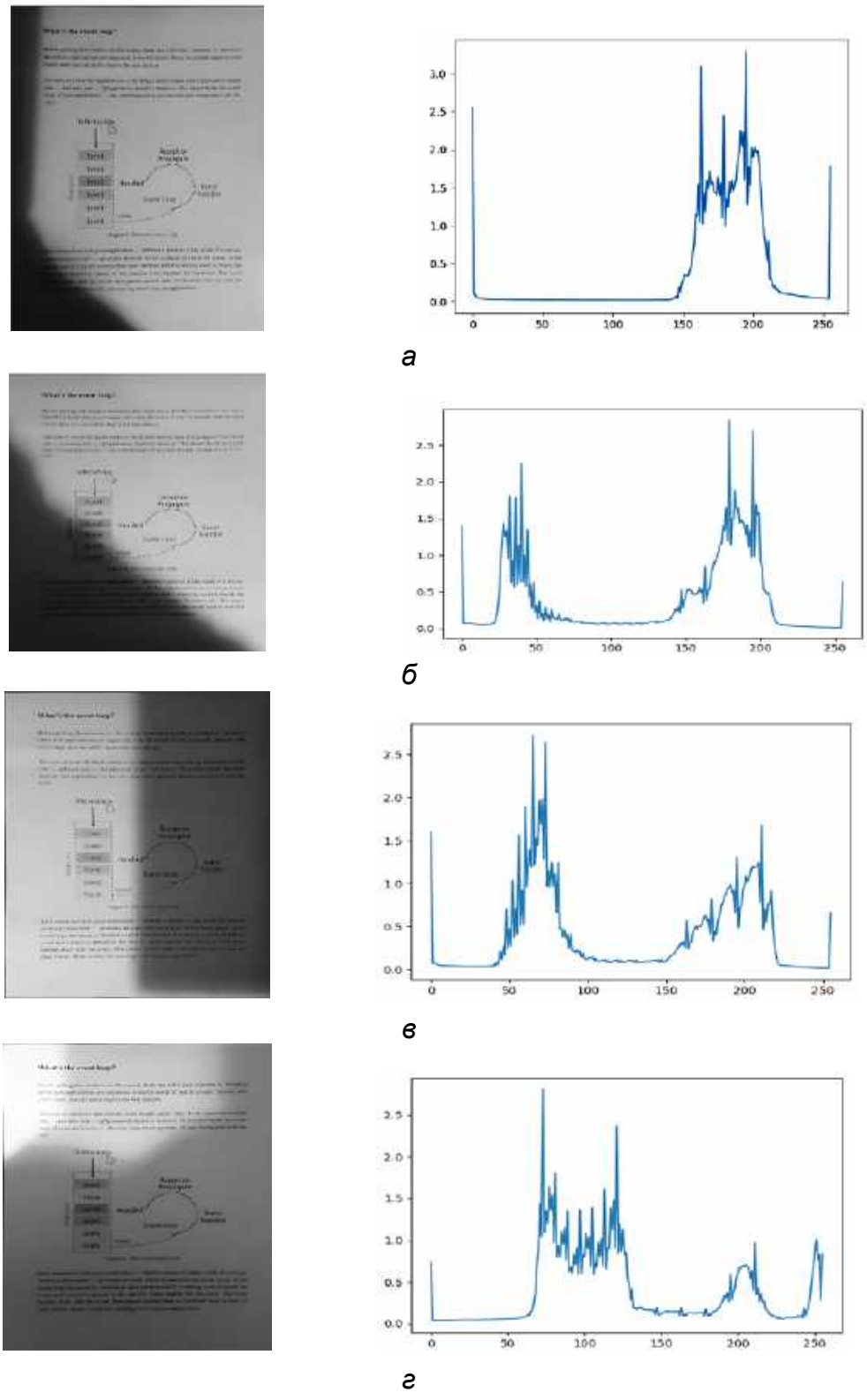
Як видно на рис. 2.26, вибір порога відіграє головну роль у бінаризації. Так, за умови використання стандартного порога 127 точність розпізнавання знизилася до 94,68 %, а за умови порога 160 − до 58,36 %. За умови використання порога 100 точність, навпаки, збільшилася до 99,71 %. Під час бінаризації Otsu поріг визначається автоматично, і, як видно з результатів, точність розпізнавання збільшилася до 99,71 %. Таку саму точність було отримано з використанням методів адаптивної бінаризації.

З експерименту можна зробити висновок, що використання стандартної порогової бінаризації не є раціональним, оскільки потребує індивідуального настроювання для кожного зображення. Що стосується бінаризації Otsu і різних адаптивних методів, то було проведено додатковий експеримент, у якому як вихідні дані використовувалися зображення з нерівномірним освітленням областей.

Зразки зображень, отримані за різних умов освітлення об'єкта, подано на рис. 2.27. Для кожного із зображень побудовані нормовані гістограми розподілу яскравості, які чітко характеризують умови освітлення сцени.

На рис. 2.28 зображено результати бінаризації зразків зображень, виконаних для різних умов освітлення. Кожне із цих зображень було оброблено методами бінаризації Otsu, адаптивного середнього й адаптивного Гаусса. З рис. 2.28 видно, що використання бінаризації Otsu призводить до зафарбовування чорним кольором усіх затінених областей вихідного зображення, що також веде до зниження точності розпізнавання. Обидва методи адаптивної бінаризації, навпаки, показали стабільну продуктивність у розв’язанні проблеми нерівномірного освітлення, наявного в перших трьох зразках зображення (див. рис. 2.28, *а*, *б*, *в*). Це

показано на рис. 2.28 і 2.29, де зберігається значення точності розпізнавання в межах 99 %. На четвертому зображенні (див. рис. 2.28, *г*) можна побачити зниження точності розпізнавання до 93 % через надмірну яскравість фрагментів тексту після бінаризації.



а

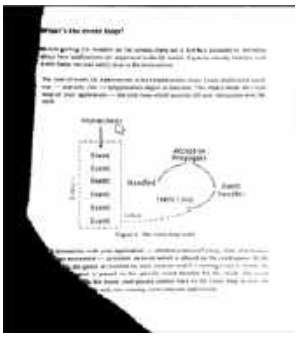
б

в

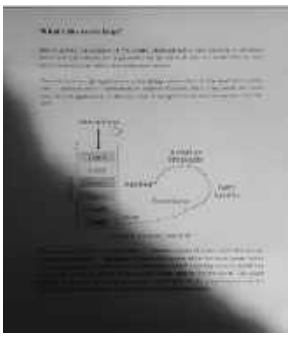
г

Рис. 2.27. Приклади знімків, зроблених за різних умов освітлення:
а − рівномірне освітлення; *б* − нижній лівий кут зображення затінений;
в − права частина зображення в тіні; *г* − майже 75 % тексту в тіні

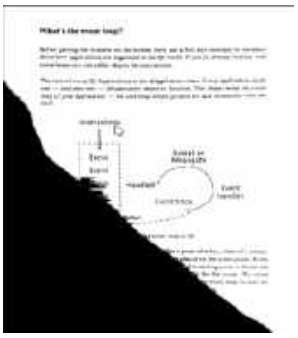
Початкове зображення



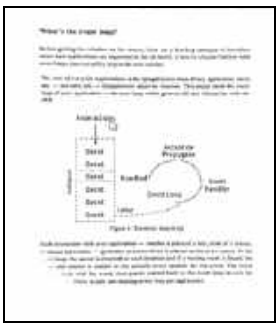
Бінаризація Otsu



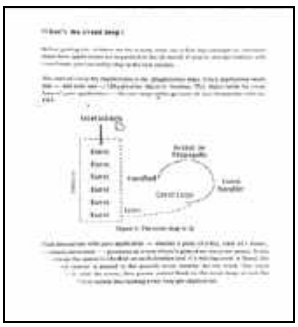
Початкове зображення



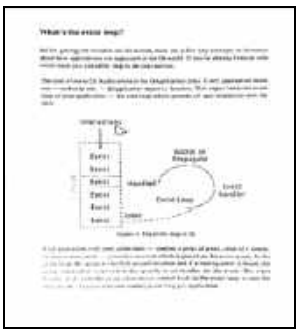
Бінаризація Otsu



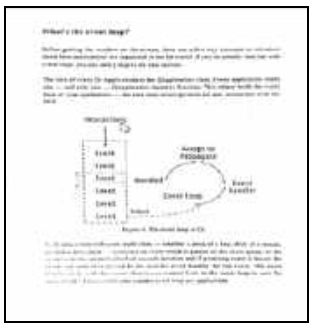
Адаптивна бінаризація (mean)



Адаптивна бінаризація (gaussian)



Адаптивна бінаризація (mean)

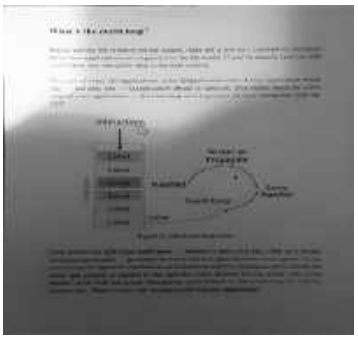


Адаптивна бінаризація (gaussian))

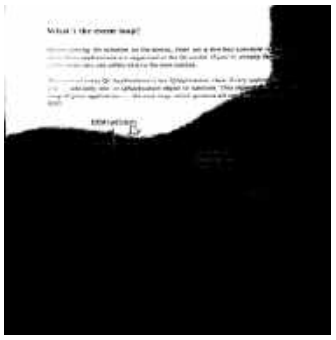
а *б*

Початкове зображення

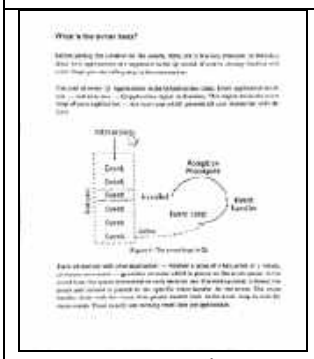
Бінаризація Otsu



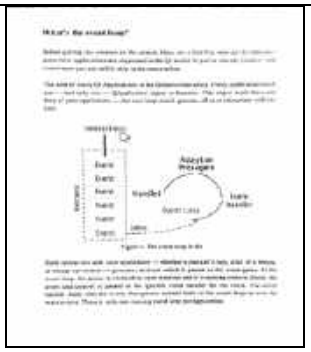
Початкове зображення



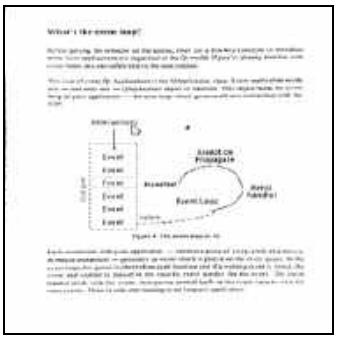
Бінаризація Otsu



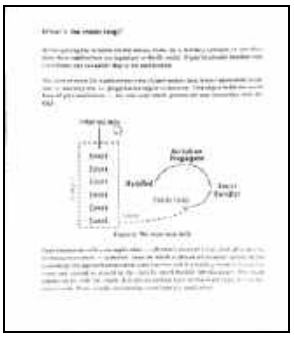
Адаптивна бінаризація (mean)



Адаптивна бінаризація (gaussian)



Адаптивна бінаризація (mean)



Адаптивна бінаризація (gaussian)

в *г*

Рис. 2.28. Результати застосування різних технік бінаризації для зразків зображень із різним освітленням: *а* − зразок 1, *б* − зразок 2, *в* − зразок 3, *г* − зразок 4

Аналіз результатів показує, що метод бінаризації з адаптивним пороговим середнім можна назвати найкращим, оскільки за допомогою цього методу досягається найвища точність розпізнавання незалежно від умов освітлення зображення.

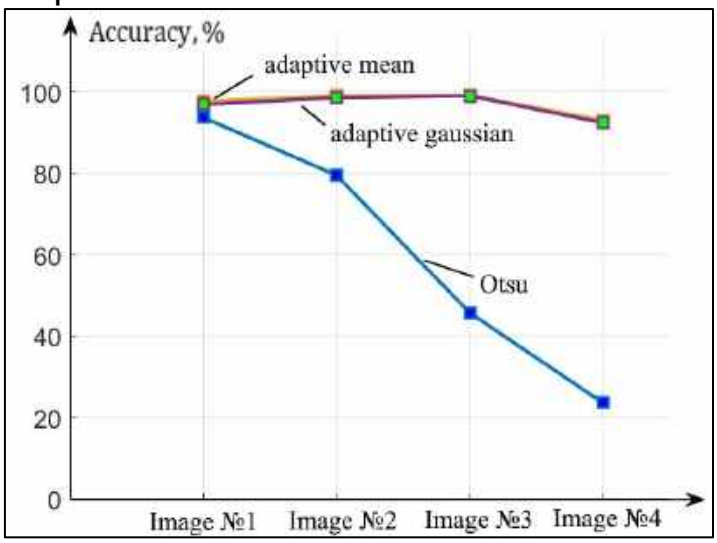


Рис. 2.29. Порівняльна характеристика ефективності методів бінаризації

2.3.5. Ерозія зображення

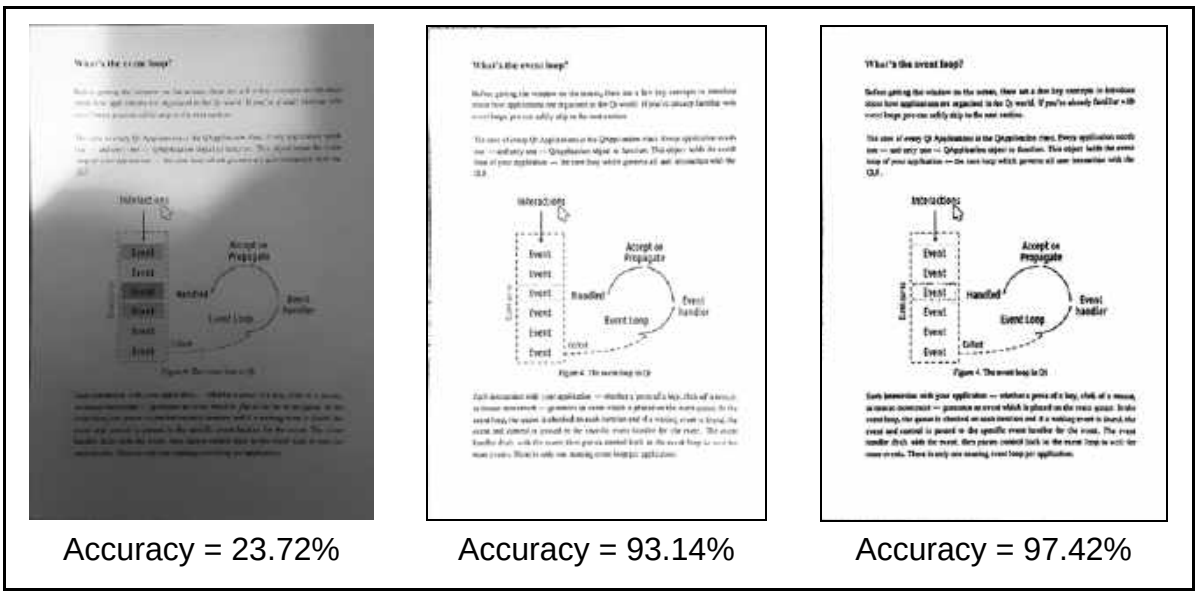
Проблему зниження контрасту зображення після адаптивної бінаризації (наприклад, рис. 2.28, *г*) можна розв'язати за допомогою особливої морфологічної операції − ерозії. Ерозія («звуження» або «розмивання») поряд із такими перетвореннями, як дилатація, розмикання і замикання є однією з основних операцій математичної морфології.

Фільтр ерозії зменшує область зображення, витончуючи пікселі, розширюючи і посилюючи світлі місця на зображенні. Суть цього перетворення полягає в тому, що небажані вкраплення і шуми «розмиваються», а великі і, відповідно, значущі ділянки зображення не змінюються. Застосування ерозії дає змогу виділити текст на зображенні після адаптивної бінаризації, що підвищує загальну точність розпізнавання тексту.

Реалізація методу для виконання ерозії наведено на рис. 2.30. Результат використання ерозії для оброблення зображень подано на рис. 2.31.

```
131     # ерозія
132     def _erosion(self, is_required):
133         if is_required:  # якщо операція необхідна
134             # створення маски
135             kernel = np.ones((self._e_ksize, self._e_ksize), np.uint8)
136             # використання функції OpenCV для ерозії
137             self._erosion_image = cv2.erode(self._binarization_image, kernel, iterations=self._e_num)
138         else:
139             self._erosion_image = self._binarization_image
```

Рис. 2.30. Реалізація методу ерозії



Accuracy = 23.72% Accuracy = 93.14% Accuracy = 97.42%

Рис. 2.31. Використання ерозії для підвищення точності розпізнавання

З огляду на результати зображень на рис. 2.31 можна зробити висновок, що використання адаптивної бінаризації (середнього) дає змогу розв'язати проблему нерівномірного освітлення зображення, а у випадках зі зниженим контрастом можна застосувати ерозію як завершальний етап попередньої підготовки.

2.3.6. Повний цикл попереднього оброблення зображень

У підрозд. 2.3.1–2.3.5 було подано окремі етапи попереднього оброблення зображень, де основна увага приділялася безпосередньо методам класу `PreProcessing` та окремим дослідженням впливу окремих етапів на точність розпізнавання. Але при цьому не було зазначено досить важливого аспекту роботи кожного з методів, а саме − параметри для виконання кожної з операцій. Усі параметри визначаються в конструкторі класу (рис. 2.32) під час створення екземпляра класу за допомогою аргумента `settings`, що являє собою повний список усіх можливих параметрів.

У цьому розділі не буде описуватися окремо кожен із параметрів, проте буде подано файл зі всіма значеннями за замовчуванням (рис. 2.33), який використовується для функціонування роботи класу. У разі потреби для більш детального ознайомлення з параметрами можна переглянути офіційну документацію бібліотеки OpenCV [1], у якій описані всі використовувані в підрозд. 2.3.1–2.3.5 функції з відповідними параметрами.

Останній із методів цього класу дає змогу виконувати повний цикл попереднього оброблення вихідного зображення відповідно до всіх параметрів, переданих конструктору класу. Його реалізацію наведено на рис. 2.34, а на рис. 2.35 – графік точності розпізнавання вихідного зображення відповідно до кожного з етапів оброблення.

```python
# клас для попередньої обробки зображення
class PreProcessing:

    # конструктор
    def __init__(self, image, settings):
        # атрибути класу
        # зчитування налашувань
        self._f_type = settings[0]
        self._g_ksize = int(settings[1])
        self._g_sigmax = int(settings[2])
        self._g_sigmay = int(settings[3])
        self._m_ksize = int(settings[4])
        self._b_d = int(settings[5])
        self._b_sigma_color = int(settings[6])
        self._b_sigma_space = int(settings[7])
        self._hf_type = settings[8]
        self._b_type = settings[9]
        self._s_thresh = int(settings[10])
        self._s_max = int(settings[11])
        self._s_type = settings[12]
        self._o_max = int(settings[13])
        self._0_type = settings[14]
        self._a_type1 = settings[15]
        self._a_type2 = settings[16]
        self._a_max = int(settings[17])
        self._a_ksize = int(settings[18])
        self._a_c = int(settings[19])
        self._e_ksize = int(settings[20])
        self._e_num = int(settings[21])
        # початкове зображення для обробки
        self._original_image = image.copy()
        # зображення у відтінках сірого
        self._grayscale_image = cv2.cvtColor(self._original_image, cv2.COLOR_BGR2GRAY)
        # зображення після фільтрації шумів
        self._filtration_image = None
        # зображення після високочастотної фільтрації
        self._hf_filtration_image = None
        # зображення після бінаризації
        self._binarization_image = None
        # зображення після ерозії
        self._erosion_image = None
```

Рис. 2.32. Конструктор класу `PreProcessing`

Отже, виконання повного циклу попереднього оброблення зображення дало змогу збільшити точність розпізнавання з 80.58 % на вихідному зображенні із шумами до 99.85 %. Цей результат свідчить про високу ефективність роботи запропонованої методики попереднього оброблення і дає змогу максимізувати точність перед виконанням безпосереднього розпізнавання тексту на зображенні. При цьому слід зазначити, що кожен з етапів можна коригувати у випадку нестандартних зображень завдяки високій гнучкості параметрів класу.

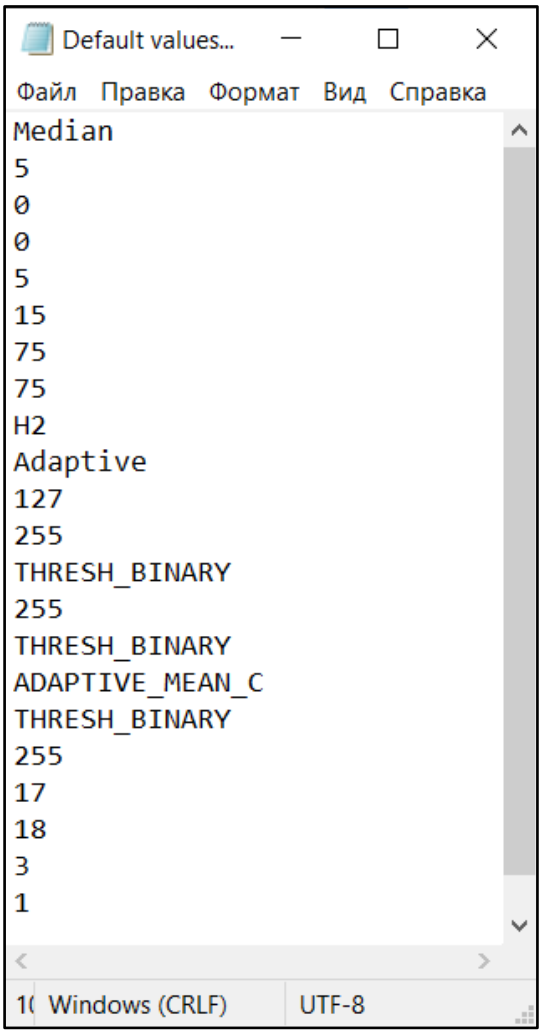


Рис. 2.33. Файл зі значеннями параметрів за замовчуванням

```
141    # виконання повного циклу попередньої обробки
142    def full_pre_processing_cycle(self, flags):
143        self._noise_filtration(flags[0])
144        self._hf_filtration(flags[1])
145        self._binarization(flags[2])
146        self._erosion(flags[3])
147        return self._filtration_image, self._hf_filtration_image, self._binarization_image, self._erosion_image
```

Рис. 2.34. Фрагмент коду для реалізації методу виконання повного циклу попереднього оброблення

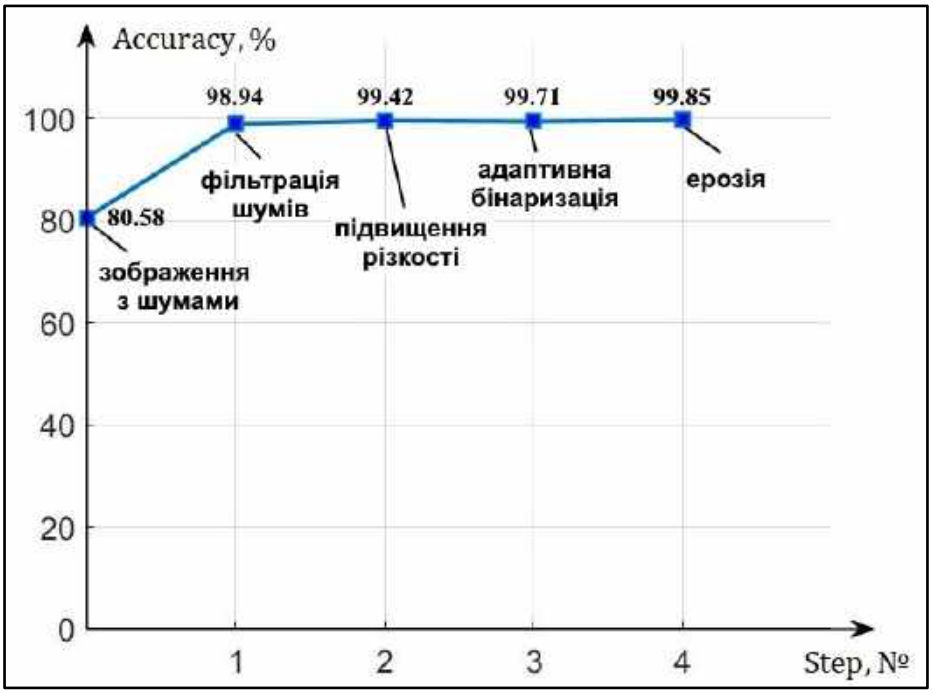


Рис. 2.35. Точность розпізнавання вихідного зображення відповідно до кожного з етапів оброблення

2.4. Сегментація та анонімізація тексту

Сегментацію розуміють як виділення із цілісного тексту окремих фрагментів прямокутної форми, що становлять найбільший інтерес для розпізнавання. Ця операція не є обов'язковою, але в разі потреби дає змогу користувачеві в інтерактивному режимі сконцентрувати увагу розпізнавача текстів тільки на окремих найбільш важливих фрагментах зображення.

Анонімізація являє собою виділення прямокутних фрагментів тексту, які в подальшому будуть надійно приховані під час розпізнавання. Ця операція дає змогу розв’язати проблеми як збереження конфіденційності інформації, так і приховування фрагментів тексту, у яких немає корисної інформації під час розпізнавання.

Для цих операцій було розроблено окремий клас `SegmentationClass`. Серед бібліотек використовуються `OpenCV` та `Imutils` (рис. 2.36).

```
1 # підключення необхідних бібліотек
2 import cv2
3 import imutils
```

Рис. 2.36. Бібліотеки для класу `SegmentationClass`

До класу належать такі компоненти:

1. Конструктор із набором атрибутів для обміну інформацією між окремими методами класу (рис. 2.37).

```
7  class SegmentationClass:
8
9      # конструктор класу
10     def __init__(self, image, color_image, fixed_height=750):
11         # атрибути класу
12         # початкове зображення
13         self._orig_image = image.copy()
14         # початкове зображення (кольорове)
15         self._color_image = color_image.copy()
16         # фіксоване значення висоти зображення при відображенні
17         self._fixed_height = fixed_height
18         # початкове зображення з фіксованою висотою
19         self._orig_image_fixed = imutils.resize(self._orig_image, height=self._fixed_height)
20         self._color_image_fixed = imutils.resize(self._color_image, height=self._fixed_height)
21         # поточне зображення з фіксованною висотою
22         self._current_fixed = self._orig_image_fixed.copy()
23         self._current_color_fixed = self._color_image_fixed.copy()
24         # початкова точка прямокутника при сегментації
25         self._start_point = (0, 0)
26         # кінцева точка прямокутника при сегментації
27         self._end_point = (0, 0)
28         # список всіх відомих сегментів (для зображення фіксованої висоти)
29         self._segment_massive = []
30         # список всіх відомих анонімних участків зображення
31         self._anon_massive = []
32         # список всіх відомих зображень для збереження
33         self._crop_massive = []
34          # флаг для малювання прямокутників при виділенні сегментів
35         self._flag = False
36         # режим роботи для обробника подій ('Segmentation' , 'Anonymization')
37         self._mode = 'Segmentation'
38         # розрахунок відношення висоти початкового зображеня до фіксованого значення
39         self._ratio = self._orig_image.shape[0] / self._fixed_height
```

Рис. 2.37. Конструктор класу `SegmentationClass`

2. Допоміжний метод `_image_rendering()`, який буде створювати прямокутні геометричні примітиви для окремих фрагментів зображення (див. рис. 2.38). Після перевірки режиму роботи створюється копія поточного зображення фіксованої висоти (рядки 43–44). Далі для кожного збереженого сегмента в списку `_segment_massive` створюється прямокутник одиничної товщини (рядки 46–48), а для списку `_anon_massive` – зафарбований прямокутник сірого кольору (рядки 50–52). Після модифікацій отримане зображення демонструється користувачу (рядок 54). Можна побачити, що метод має третій режим роботи, функціонал якого не буде описуватися в цьому розділі.

```
41    # метод для відрисовки сегментів та анонімних участків
42    def _image_rendering(self):
43        if self._mode == 'Segmentation' or self._mode == 'Anonymization':
44            self._current_fixed = self._orig_image_fixed.copy()
45            # для кожного набору точок масиву
46            for segment in self._segment_massive:
47                # малюємо прямокутник одиничної товщини
48                cv2.rectangle(self._current_fixed, segment[0], segment[1], (0, 0, 255), 2)
49            # для кожного набору точок масиву
50            for anon in self._anon_massive:
51                # малюємо зафарбований прямокутник білого кольору
52                cv2.rectangle(self._current_fixed, anon[0], anon[1], (230, 230, 230), -1)
53            # оновлюємо головне вікно для демонстрації
54            cv2.imshow(self._mode + " Window", self._current_fixed)
55        elif self._mode == 'Picture croping':
56            self._current_color_fixed = self._color_image_fixed.copy()
57            # для кожного набору точок масиву
58            for croped in self._crop_massive:
59                # малюємо зафарбований прямокутник білого кольору
60                cv2.rectangle(self._current_color_fixed, croped[0], croped[1], (0, 0, 255), 2)
61            # оновлюємо головне вікно для демонстрації
62            cv2.imshow(self._mode + " Window", self._current_color_fixed)
```

Рис. 2.38. Фрагмент коду для реалізації методу `_image_rendering()`

3. Метод `clear_all()`для видалення накопиченої інформації з інформаційних масивів (рис. 2.39).

```
103    # метод для очищення масивів даних та повернення початкового зображення
104    def clear_all(self):
105        self._current_fixed = self._orig_image_fixed.copy()
106        self._segment_massive.clear()
107        self._anon_massive.clear()
108        self._crop_massive.clear()
109        return self._orig_image.copy()
```

Рис. 2.39. Фрагмент коду для реалізації методу `clear_all()`

4. Метод `set_mode()`для заміни поточного режиму роботи класу (рис. 2.40).

```
111     # метод для зміни режиму роботи
112     def set_mode(self, mode="Segmentation"):
113         self. mode = mode
```

Рис. 2.40. Фрагмент коду для реалізації методу `set_mode()`

5. Обробник подій `_click_and_crop()`, запуск якого дає змогу визначити під час сегментації або анонімізації координати кутових точок фрагмента в реальному часі (рис. 2.41).

В обробнику відстежуються такі маніпуляції:

- натискання лівої кнопки миші, що приводить до підняття логічного прапора `_flag` і збереження початкових координат (рядки 67–70);
- переміщення курсора, при якому перевіряється значення прапора `_flag`; у разі значення «істина» курсор буде створювати тимчасовий прямокутник, який відображає координати початкової (`_start_point`) і поточної (`tmp_end`) точок (рядки 72–85);
- відпускання лівої кнопки миші; тут прапор `_flag` повертає початкове значення, а кінцева координата `_end_point` зберігається (рядки 87–90); кінцеві координати `_start_point` і `_end_point` будуть збережені у відповідних масивах поточного режиму (рядки 91–99).

Після збереження цих координат викликається допоміжний метод `_image_rendering()` для відтворення геометричних примітивів відповідно до оновлених масивів даних (рядок 101).

6. Основний метод `image_pinning()`, у якому створюється іменоване вікно з текстовим зображенням для інтерактивної роботи (рис. 2.42 та 2.43).

Після створення та відображення вікна, до якого прикріплений обробник подій `_click_and_crop()` (рядки 118–122) починається робочий цикл методу, під час якого користувачеві надаються такі можливості:

- натискання клавіші «`r`» (оновити) повертає вихідне зображення й очищає всі інформаційні масиви (рядки 124–136);
- натискання клавіші «`b`» (назад) скасовує останню виконану операцію (рядки 138–145);
- натискання клавіші «`enter`» (вихід) закриває інтерактивне вікно, перераховує всі координати в інформаційних масивах за розміром вихідного зображення і повертає результат методу (рядки 147–174).

Метод повертає зображення `image` з усіма виділеними сегментами та прихованими фрагментами після анонімізації. Окрім цього, також повертається масив `roi_massive`, який містить окремо кожен із виділених користувачем сегментів.

Також слід зазначити, що для кожного окремого зображення текстового документа можна використовувати як поєднання сегментації з анонімізацією, так і будь-яку із цих операцій окремо. Навіть можна

пропустити цей етап оброблення. У цьому випадку встановлюється режим за замовчуванням, при якому розпізнається весь текст зображення.

```
    # метод для обробки подій
    def _click_and_crop(self, event, x, y, flags, param):
        # при натисканні лівої кнопки миші
        if event == cv2.EVENT_LBUTTONDOWN:
            self._flag = True  # підіймаємо відповідний флаг
            # запам'ятовуємо початкову точку
            self._start_point = (int(x), int(y))
        # при пересуванні курсору
        elif event == cv2.EVENT_MOUSEMOVE:
            # перевірка дійсності флага
            if self._flag:
                # тимчасова кінцева точка
                tmp_end = (int(x), int(y))
                # отримуємо копію поточного зображення
                if self._mode == 'Segmentation' or self._mode == 'Anonymization':
                    tmp_image = self._current_fixed.copy()
                elif self._mode == 'Picture croping':
                    tmp_image = self._current_color_fixed.copy()
                # малюємо тимчасовий прямокутник
                cv2.rectangle(tmp_image, self._start_point, tmp_end, (0, 0, 0), 1)
                # оновлюємо головне вікно для демонстрації
                cv2.imshow(self._mode+" Window", tmp_image)
        # при віджиманні лівої кнопки миші
        elif event == cv2.EVENT_LBUTTONUP:
            self._flag = False  # опускаємо флаг
            # запам'ятовуємо кінцеву точку
            self._end_point = (int(x), int(y))
            if self._mode == 'Segmentation':  # при режимі сегментації
                # записуємо початкову і кінцеву точки у відповідний масив
                self._segment_massive.append([self._start_point, self._end_point])
            elif self._mode == 'Anonymization':  # при режимі анонімізації
                # записуємо початкову і кінцеву точки у відповідний масив
                self._anon_massive.append([self._start_point, self._end_point])
            elif self._mode == 'Picture croping':  # при режимі збереження зображення
                # записуємо початкову і кінцеву точки у відповідний масив
                self._crop_massive.append([self._start_point, self._end_point])
            # викликаємо функцію для відмальовування геометричних примітивів
            self._image_rendering()
```

Рис. 2.41. Фрагмент коду для реалізації обробника подій `_click_and_crop()`

```
115     # отримання зображення для сегментації/анонімізації
116     def image_pinning(self):
117         # створюємо іменоване вікно для зображення
118         cv2.namedWindow(self._mode+" Window")
119         # закріплюємо за іменованим вікном обробник
120         cv2.setMouseCallback(self._mode + " Window", self._click_and_crop)
121         # відображення зображення
122         self._image_rendering()
123         # робочий цикл
124         while True:
125             key = cv2.waitKey(1)
126             # при натисканні клавіши 'r' повертаємо початкове зображення
127             if key == ord("r"):
128                 self._current_fixed = self._orig_image_fixed.copy()
129                 self._current_color_fixed = self._color_image_fixed.copy()
130                 if self._mode == 'Segmentation':
131                     self._segment_massive.clear()
132                 elif self._mode == 'Anonymization':
133                     self._anon_massive.clear()
134                 elif self._mode == 'Picture croping':
135                     self._crop_massive.clear()
136                 self._image_rendering()
```

Рис. 2.42. Фрагмент коду для реалізації першої частини методу `image_pinning()`

```
137             # при натисканні  клавіши 'b' видаляємо останню виконану операцію
138             elif key == ord("b"):
139                 if self._mode == 'Segmentation' and len(self._segment_massive) > 0:
140                     self._segment_massive.pop()
141                 elif self._mode == 'Anonymization' and len(self._anon_massive) > 0:
142                     self._anon_massive.pop()
143                 elif self._mode == 'Picture croping' and len(self._crop_massive) > 0:
144                     self._crop_massive.pop()
145                 self._image_rendering()
146             # при натисканні  клавіши 'e' закриваємо робоче вікно
147             elif key == 13:
148                 image = self._orig_image.copy()
149                 roi_massive = []  # список всіх відомих сегментів (для початкового зображення)
150                 # масштабування точок в масивах під початкове зображення
151                 for anon in self._anon_massive:
152                     tmp_start = (int(anon[0][0]*self._ratio), int(anon[0][1]*self._ratio))
153                     tmp_end = (int(anon[1][0]*self._ratio), int(anon[1][1]*self._ratio))
154                     cv2.rectangle(image, tmp_start, tmp_end, (230, 230, 230), -1)
155                 if self._mode == 'Segmentation' or self._mode == 'Anonymization':
156                     for segment in self._segment_massive:
157                         tmp_start = (int(segment[0][0]*self._ratio), int(segment[0][1]*self._ratio))
158                         tmp_end = (int(segment[1][0]*self._ratio), int(segment[1][1]*self._ratio))
159                         cv2.rectangle(image, tmp_start, tmp_end, (0, 0, 255), 5)
160                         roi_massive.append(image[tmp_start[1]:tmp_end[1], tmp_start[0]:tmp_end[0]])
161                     # якщо не було виділено жодного сегменту
162                     if len(roi_massive) == 0:
163                         # записуємо у список початкове зображення
164                         roi_massive.append(image)
165                 elif self._mode == 'Picture croping':
166                     color_image = self._color_image.copy()
167                     for croped in self._crop_massive:
168                         tmp_start = (int(croped[0][0]*self._ratio), int(croped[0][1]*self._ratio))
169                         tmp_end = (int(croped[1][0]*self._ratio), int(croped[1][1]*self._ratio))
170                         cv2.rectangle(image, tmp_start, tmp_end, (0, 0, 255), 5)
171                         roi_massive.append(color_image[tmp_start[1]:tmp_end[1], tmp_start[0]:tmp_end[0]])
172                 # закриваємо вікно
173                 cv2.destroyAllWindows()
174                 return image, roi_massive
```

Рис. 2.43. Фрагмент коду для реалізації другої частини методу `image_pinning()`

Результати роботи модуля, що включають приклади використання всіх перелічених операцій, показано на рис. 2.44. Метод дає змогу виконувати сегментацію та анонімізацію в реальному часі з подальшим збереженням отриманих результатів.

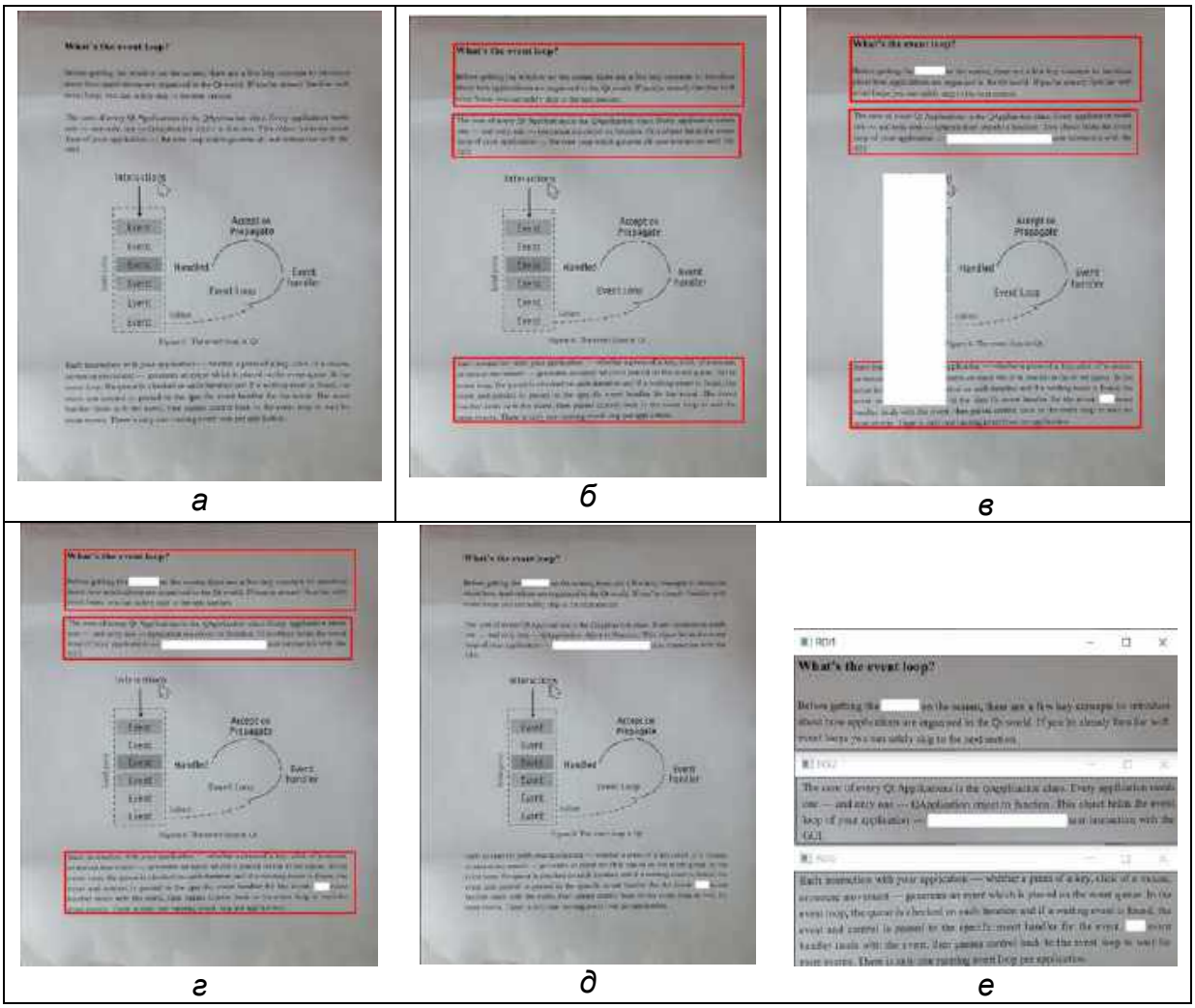


Рис. 2.44. Результати роботи `Segmentation_module`:
а − вихідне зображення; *б* − виділення окремих фрагментів; *в* − зміна режиму й анонімізація декількох (а саме чотирьох) фрагментів; *г* − скасування останньої операції; *д* − остаточне зображення; *е* − повернуті фрагменти тексту

2.5. Оптичне розпізнавання тексту з використанням Python і Tesseract

Оптичне розпізнавання тексту зазвичай розуміють як електронний переказ рукописних, машинописних або друкованих зображень у текстові дані, які використовуються для подання символів у комп'ютері (наприклад, у текстовому редакторі).

Зараз найбільш поширені так звані інтелектуальні системи, що розпізнають більшість шрифтів із високим ступенем точності. Tesseract є прикладом таких інтелектуальних систем.

Tesseract − це рушій оптичного розпізнавання символів (OCR) з відкритим вихідним кодом, який використовує нейронні мережі для пошуку і розпізнавання тексту на зображеннях. З версії 4.0 рушій Tesseract почав

використовувати архітектуру довготривалої короткостроковій пам'яті (LSTM) для рекурентних нейронних мереж.

Як зазначалося раніше, для спільного використання мови програмування Python і рушія Tesseract існує спеціальна бібліотека `pytesseract` [1], на основі якої було розроблено клас `TessOcr` для розпізнавання тексту на зображеннях і подальшого відображення результатів. Серед бібліотек, окрім `pytesseract`, використовуються `OpenCV`, `os`, `Pillow` (рис. 2.45). Реалізацію конструктора наведено на рис. 2.46.

```
1 # підключення необхідних бібліотек
2 import pytesseract
3 import cv2
4 import os
5 from PIL import Image
```

Рис. 2.45. Бібліотеки для класу `TessOcr`

```
8  # клас для розпізнавання тексту
9  class TessOcr:
10
11     # методи класу
12     # конструктор класу
13     def __init__(self, roi_massive, language):
14         # атрибути класу
15         # шлях до Tesseract на пристрої
16         self._tesseract_path = r'Tesseract-OCR\tesseract.exe'
17         # сегменти для розпізнавання
18         self._roi_massive = roi_massive
19         # мова тексту для розпізнавання
20         self._language = language
21         # шлях до Tesseract на пристрої
22         pytesseract.pytesseract.tesseract_cmd = self._tesseract_path
```

Рис. 2.46. Конструктор класу `TessOcr`

У розробленому класі використовується метод `preview()`, який дає змогу розпізнати текст на всіх окремих сегментах вихідного масиву `_roi_massive` та об'єднати їх в єдиний текст. Реалізацію методу подано на рис. 2.47.

Повний алгоритм роботи методу передбачає такі етапи:

1. Створення змінної для подальшого збереження повного тексту (рядок 27).
2. Для кожного сегмента тексту створюється тимчасовий файл зображення у форматі `jpg` (рядки 30–32).
3. Завантаження `jpg` як файлу бібліотеки `Pillow`, виконання розпізнавання і видалення тимчасового файлу (рядки 35–36).
4. Видалення пустих рядків у розпізнаному фрагменті тексту та додавання його до повного тексту (рядки 38–40).

5. Повернення повного тексту `full_text` як результат роботи методу (рядок 42).

```
24      # метод для розпізнавання тексту (без збереження)
25      def preview(self):
26          # змінна для тексту з усіх сегментів
27          full_text = ""
28          # Запис кожного із зображень на диск як тимчасового файла
29          # для подальшого розпізнавання
30          for image in self._roi_massive:
31              filename = "tmp_file.jpg".format(os.getpid())
32              cv2.imwrite(filename, image)
33              # загрузка зображення як PIL/Pillow , використання OCR
34              # й видалення тимчасового файла
35              segment_text = pytesseract.image_to_string(Image.open(filename), lang=self._language)
36              os.remove(filename)
37              # прибираємо пусті строки
38              for line in segment_text.splitlines():
39                  full_text += line + ' '
40              full_text += "\n"
41          # повертаємо повний текст як результат роботи
42          return full_text
```

Рис. 2.47. Фрагмент коду для реалізації методу `preview()`

Окрім наведеного методу розпізнавання, у класі також було реалізовано метод `get_params()` для визначення додаткових параметрів тексту під час розпізнавання за допомогою ресурсів бібліотеки `pytesseract`. Реалізацію методу показано на рис. 2.48.

```
44      # метод для визначення параметрів тексту
45      def get_params(self):
46          filename = "tmp_file.jpg".format(os.getpid())
47          cv2.imwrite(filename, self._roi_massive[0])
48          # загрузка зображення як PIL/Pillow , використання OCR
49          # й видалення тимчасового файла
50          params = pytesseract.image_to_osd(Image.open(filename), output_type=pytesseract.Output.DICT)
51          os.remove(filename)
52          return params
```

Рис. 2.48. Фрагмент коду для реалізації методу `get_params()`

Докладно описані в цьому розділі класи доцільно використовувати під час створення програми оптичного розпізнавання тексту. Зазвичай такі програми реалізують за модульним принципом.

3. ЗАХИСТ ІНФОРМАЦІЇ ПІД ЧАС ПЕРЕДАВАННЯ ДАНИХ У МЕРЕЖАХ ЗВ'ЯЗКУ

Необхідно пам'ятати, що при постановці завдання проєктування системи розпізнавання тексту було сформульовано базову концепцію її створення, яка передбачає реалізацію таких обов'язкових положень:

- оцінювання й облік зовнішніх факторів, що негативно впливають на результат роботи системи, та створення алгоритмів попереднього оброблення зображень із метою ефективної компенсації впливу негативних чинників;
- надання набору сервісних функцій для зручності оброблення вихідних даних, їх перегляду, перетворення та збереження в стандартних форматах, сегментація окремих фрагментів тексту, виділення рисунків тощо;
- вживання заходів захисту інформації після розпізнавання текстових документів; передбачення ефективного кодування даних і приховання фактів передавання інформації незахищеними каналами зв'язку за допомогою методів стеганографії;
- проведення комплексного оцінювання ефективності роботи системи з набором найбільш важливих показників якості.

Зазначимо, що найважливішим є розроблення методів і алгоритмів захисту під час передавання конфіденційних даних, отриманих у процесі розпізнавання текстових документів у мережі «Інтернет». Важливість проблеми зумовлена тим, що в більшості відомих систем OCR заходи захисту інформації або недостатні, або не передбачені зовсім. Крім цього, немає загальноприйнятого підходу до методів захисту інформації. Тому доцільно розглянути це завдання докладно.

3.1. Структура сучасної системи розпізнавання тексту

У межах прийнятої концепції може бути створено два види систем оптичного розпізнавання текстових документів − без інформаційного захисту результатів розпізнавання та захищених систем для передавання конфіденційних даних.

Перший варіант структури системи розпізнавання тексту наведено на рис. 3.1. Система містить архів фотографій текстових документів, блок попереднього оброблення даних для усунення геометричних спотворень, фільтрації та підвищення контрастності фотознімків, їх бінаризації перед розпізнаванням. Тут також закладено опцію інтерактивного виділення рисунків, діаграм і таблиць із можливістю їх збереження в стандартному форматі.

У системі передбачено набір необхідних сервісних функцій (установлення робочих параметрів, перегляд результатів проміжних перетворень, збереження в стандартних форматах результатів, їх друк тощо).

У попередніх розділах було показано, що зараз систему розпізнавання тексту доцільно створювати мовою програмування Python із використан-

ням ресурсів бібліотеки OpenCV. У цьому випадку розпізнавання зручно проводити за допомогою рушія Tesseract із використанням функціонала спеціальної бібліотеки `pytesseract`. Це дає змогу створювати складні та функціональні варіанти систем оптичного розпізнавання тексту.

Рис. 3.1. Структура системи розпізнавання текстових даних
без інформаційного захисту

Результати розпізнавання доцільно записувати в стандартному форматі `*.docx`, а зображення, що трапляються в тексті, виділяти і зберігати у форматі `*.png` на комп'ютері користувача. Потім у разі потреби файли можна передати електронною поштою іншим користувачам, але в цьому випадку захисту інформації немає. Це є основним і дуже суттєвим недоліком такої конфігурації системи розпізнавання тексту.

Для створення засобів захисту необхідно використовувати процедуру кодування секретних повідомлень і надійно маскувати факти передавання даних методами стеганографії. Загальну схему комунікацій у захищеному режимі наочно показано на рис. 3.2. Учасники процесу «Відправник – Одержувач» роблять передавання та приймання стегозображення з вкладеним у нього секретним текстом. Незначний недолік методу полягає в тому, що «Одержувач» для отримання та декодування секретної інформації повинен мати для цього відповідне програмне забезпечення. Проте цей метод завжди надійно розв’язує проблему захисту від несанкціонованого її використання третіми особами.

На рис. 3.3. показано узагальнену структурну схему передавальної частини системи розпізнавання тексту з програмними блоками для кодування текстової інформації за допомогою QR-кодів і приховування фактів передавання даних із використанням алгоритмів LSB (Least Significant Bit)-стеганографії. Крім цього, у системі використовується бібліотека зображень-обкладинок для створення файлів стеганографії з вкладеними в них

секретними повідомленнями. Періодична зміна зображень-обкладинок сприяє додатковому захисту інформації від спроб несанкціонованого доступу. Процедура обміну стеганографічними файлами стандартна.

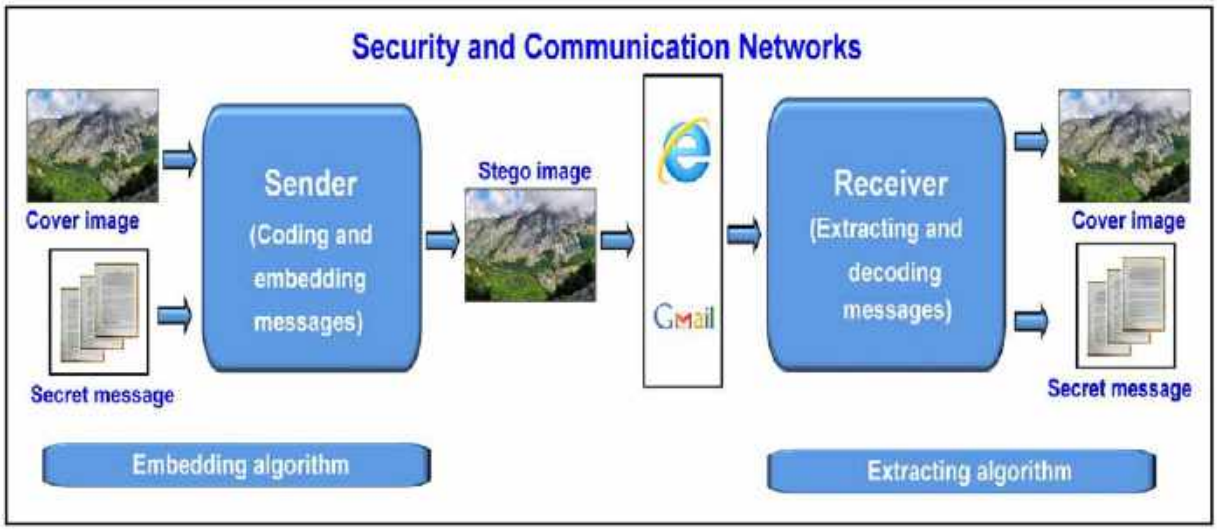


Рис. 3.2. Узагальнена схема передавання конфіденційних даних у захищеному режимі

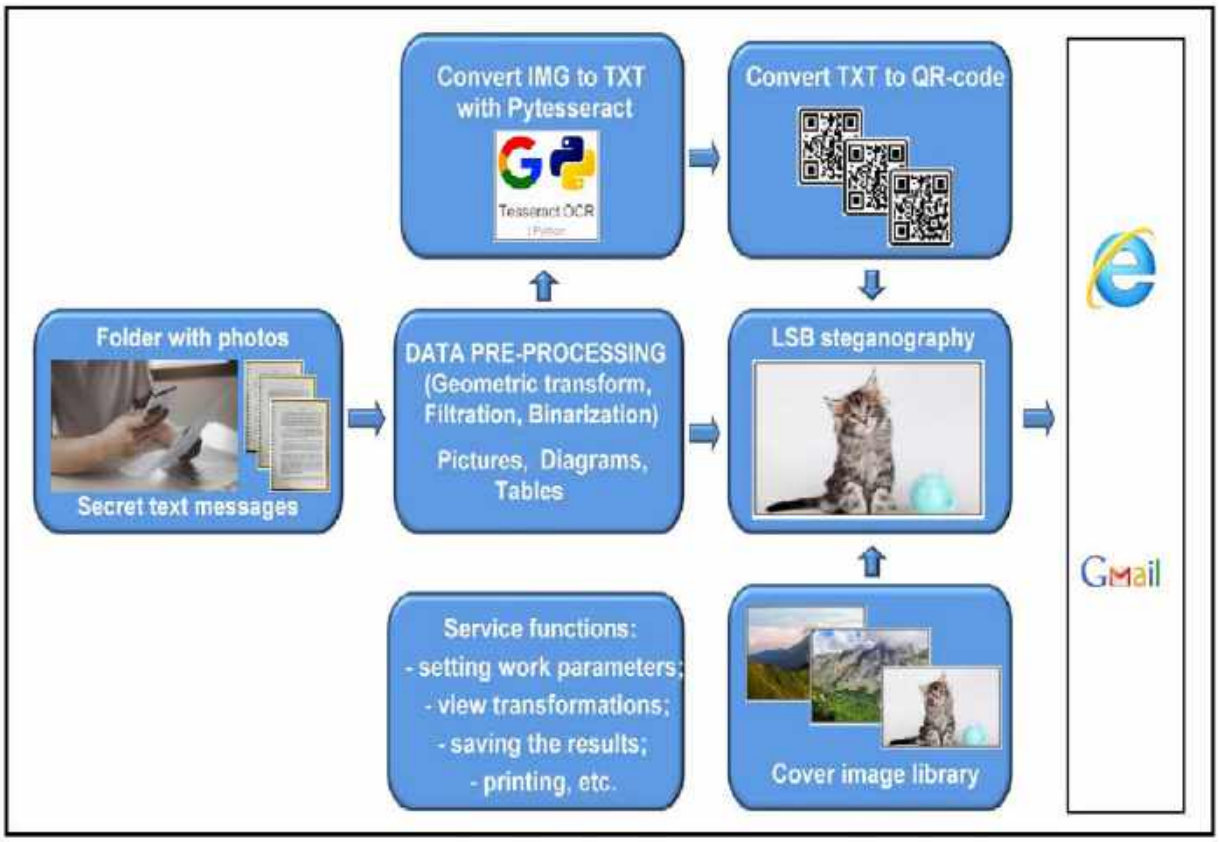


Рис. 3.3. Структура системи оптичного розпізнавання тексту із захистом отриманої інформації

3.2. Вибір методів кодування текстових даних

Захист інформації, отриманої внаслідок розпізнавання текстових документів, зазвичай складається з двох етапів – кодування даних та маскування факту передавання повідомлень у відкритих мережах.

Класичний підхід до завдання кодування текстової інформації передбачає використання методів криптографії [24, 25]. Проте не менш популярним останнім часом став метод кодування текстової інформації за допомогою QR-кодів [26, 27], що є простішим в обчисленні та дає змогу кодувати великі обсяги текстової інформації. Крім того, бінарний формат зображення QR-коду добре поєднується з форматами, що використовуються в LSB-стеганографії для прихованого передавання повідомлень. Автори пропонують комплексний метод захисту текстових даних, що базується на спільному використанні QR-кодів та алгоритмів LSB-стеганографії.

Розглянемо необхідні властивості QR-кодів докладніше. Захищене передавання текстової інформації передбачає генерацію довільної кількості QR-кодів для перетворення текстових даних у зображення для їх подальшого приховання за допомогою LSB-стеганографії. При цьому необхідно забезпечити поділ тексту на фрагменти, кожен із яких відповідає максимально можливому обсягу QR-коду. Потім згенеровані зображення QR-кодів спеціально перетворюються для подальшого виконання алгоритмів LSB-стеганографії з метою приховання в зображенні-обкладинці на місці її молодших бітів. Загальну структуру методу спільного використання QR-кодів і LSB-стеганографії показано на рис. 3.4.

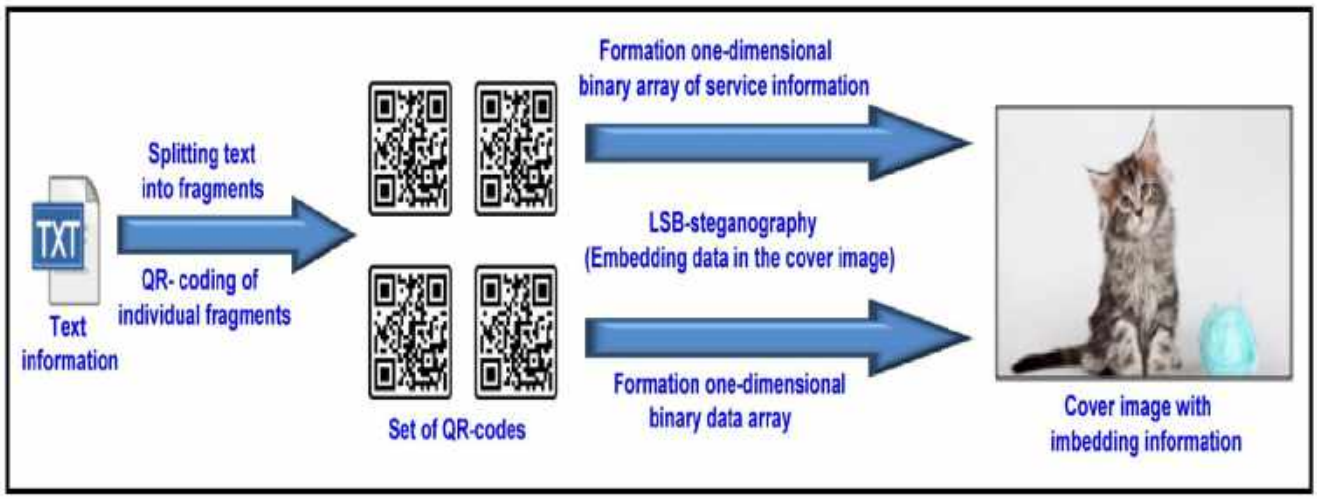


Рис. 3.4. Схема спільного використання QR-кодів та LSB-стеганографії

Розглянемо основні проблеми, що виникають під час реалізації пропонованого комбінованого методу захисту даних, та заходи для їх конструктивного розв’язання.

На першому етапі потрібно провести генерацію довільної кількості QR-кодів для перетворення текстових даних на зображенні для їх подальшого приховання за допомогою LSB-стеганографії. Для цього важливо правильно оцінити максимально допустиму ємність окремого QR-коду та визначити його найважливіші параметри.

Слід нагадати, що процес генерації QR-коду поділяється на кілька етапів: кодування даних; додавання службової інформації та її заповнення;

поділ інформації на блоки; створення байтів корекції; об'єднання блоків; розміщення інформації на QR-коді.

Найважливіші етапи – додавання службової інформації. Для цього треба визначити рівень корекції: що вище цей рівень, то вище допустимий рівень пошкодження зображення і тим менше інформації за умови рівного розміру. Існує чотири рівні корекції: L (припустимо максимум 7 % пошкоджень), M (15 %), Q (25 %) та H (30 %). Зазвичай використовують рівень M.

Ще одна важлива властивість QR коду – його версія (чим версія новіша, тим більший розмір). Усього існує 40 версій. Номер версії залежить від кількості інформації, що кодується, і від рівня корекції. Від вибраної версії залежить вихідний розмір матриці, а також максимально можлива кількість символів, які можна закодувати. Мінімальною версією є 1, а максимальною − 40. Відмінність між ними можна побачити в табл. 3.1, у якій параметри проміжних версій не зазначено.

Під час експериментів було визначено такі особливості роботи з QR-кодами:

1. У процесі генерації QR-кодів текстова інформація попередньо перетворюється на бінарний формат, якому відповідає стовпець «Binary» в табл. 3.1. Отже, максимально можлива кількість символів тексту, що кодуються, дорівнює 2953 для 40 версії QR-коду за умови рівня L.

2. Перетворення на бінарний формат виконується за стандартами UTF-8. Це означає, кожен символ тексту містить від 1 до 4 байтів інформації. Для контролю наповненості QR-кодів кожен символ тексту окремо перетворюється на бінарний формат із подальшою перевіркою можливості переповнення.

3. У разі повного заповнення QR-коду іноді виникають помилки під час декодування інформації. Тому було ухвалено рішення додати коригувальну складову обсягом 20 байтів. Це значення було отримано експериментально та дало змогу усунути помилки в процесі декодування.

4. Для корекції помилок QR-кодів було вибрано рівня L, що дає змогу використовувати найбільший обсяг кодованої інформації. Під час кодування та декодування QR-кодів не було помічено жодних помилок. Тому немає необхідності підвищення якості корекції.

Таблиця 3.1

Інформаційна ємність QR-кодів

Version	Modules	ECC level	Data bits (mixed)	Numeric	Alphanumeric	Binary
1	21x21	L	152	41	25	17
		M	128	34	20	14
		Q	104	27	16	11
		H	72	17	10	7
40	177x177	L	23648	7089	4296	2953
		M	18672	5596	3391	2331
		Q	13328	3993	2420	1663
		H	10208	3057	1852	1273

Для реалізації роботи з QR-кодами було розроблено клас `QrCodeManager`. Цей клас дає змогу виконувати як пряме перетворення текстової інформації в послідовність QR-кодів, так і зворотне перетворення з кодів у текст. При цьому є можливість вибирати рівні корекції та версію QR-кодів. Конструктор класу разом із необхідними бібліотеками подано на рис. 3.5.

```
1  import math
2  import qrcode
3  import numpy as np
4  from pyzbar import pyzbar
5  import cv2
6
7
8  class QrCodeManager:
9
10     # конструктор класу
11     def __init__(self):
12         self.version = None
13         self.error_correction = qrcode.constants.ERROR_CORRECT_L
14         self.box_size = 10
15         self.border = 4
16         self.correction = 20
17         self.maxBytes = 2953 - self.correction
```

Рис. 3.5. Бібліотеки та конструктор класу `QrCodeManager`

Методи класу можна умовно поділити на три категорії: методи настроювання, кодування та декодування. До першої категорії належать методи `set_error_correction()` для встановлення необхідного рівня корекції та `set_qr_version()` для встановлення бажаної версії QR-коду. Реалізацію цих методів наведено на рис. 3.6.

```
20     # встановлення методу корекції помилок
21     def set_error_correction(self, c_type):
22         if c_type == 'L':
23             self.error_correction = qrcode.constants.ERROR_CORRECT_L
24             self.maxBytes = 2953 - self.correction
25         elif c_type == 'M':
26             self.error_correction = qrcode.constants.ERROR_CORRECT_M
27             self.maxBytes = 2331 - self.correction
28         elif c_type == 'Q':
29             self.error_correction = qrcode.constants.ERROR_CORRECT_Q
30             self.maxBytes = 1663 - self.correction
31         elif c_type == 'H':
32             self.error_correction = qrcode.constants.ERROR_CORRECT_H
33             self.maxBytes = 1273 - self.correction
34
35     # вибір бажаної версії QR-коду
36     def set_qr_version(self, version):
37         if 0 < version <= 40:
38             self.version = version
39         else:
40             raise ValueError('Wrong QR-code version!')
```

Рис. 3.6. Фрагмент коду для реалізації методів настроювання QR-кодів

Для кодування текстової інформації використовуються методи `generate_qr_code()` для формування одиничного QR-коду, `byte_analysis()` для перевірки необхідної кількості QR-кодів та `generate_multi_qr_codes()` для перетворення текстової інформації в послідовність QR-кодів. Реалізацію цих методів подано на рис. 3.7–3.9.

```
53      # формування одиничного QR-коду
54      def generate_qr_code(self, in_text):
55          # базові параметри QR коду
56          qr = qrcode.QRCode(
57              version=self.version,
58              error_correction=self.error_correction,
59              box_size=self.box_size,
60              border=self.border
61          )
62          # кодування тексту
63          qr.add_data(in_text)
64          # перетворення QR коду у зображення
65          try:
66              flag = True
67              qr.make(fit=True)
68              img = qr.make_image(fill_color="black", back_color="white").convert('RGB')
69              img = cv2.cvtColor(np.asarray(img), cv2.COLOR_RGB2BGR)
70              img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
71          except qrcode.exceptions.DataOverflowError:
72              img = None
73              flag = False
74          return img, flag
```

Рис. 3.7. Фрагмент коду для реалізації методу `generate_qr_code()`

```
42      # визначення необхідної кількості QR-кодів
43      def byte_analysis(self, in_text):
44          byte_array = bytearray(in_text, 'utf8')
45          byte_list = []
46          for byte in byte_array:
47              binary_representation = bin(byte)
48              byte_list.append(binary_representation)
49          bytes_num = len(byte_list)
50          qr_num = math.ceil(bytes_num / self.maxBytes)
51          return bytes_num, qr_num
```

Рис. 3.8. Фрагмент коду для реалізації методу `byte_analysis()`

У цьому фрагменті (див. рис. 3.7) створюється екземпляр класу `QRCode` бібліотеки `qrcode`, у який передається інформація про версію, рівень корекції та додаткові параметри розміру QR-коду (рядки 56–61). Далі відбувається кодування тексту за допомогою методу `add_data()` (рядок 63) з подальшим перетворенням отриманого QR-коду на зображення у форматі grayscale (рядки 65–73). Як результат роботи метод повертає зображення QR-коду та прапорець, що відповідає за успішність виконання перетворення (рядок 74).

Наступний фрагмент (див. рис. 3.8) використовується для визначення кількості байтів у тексті. Так, найперше текст перетворюється в масив байтів і створюється порожній масив для збереження подальшої інформації (рядки 44–45). Далі кожний байт перетворюється в бінарний формат і додається до масиву `byte_list` (рядки 46–48). Після проходження по всіх байтах отримуємо загальну кількість бінарних символів і розраховуємо необхідну кількість QR-кодів (рядки 49–51).

Метод `generate_multi_qr_codes()`(див. рис. 3.9) є основним методом категорії кодування текстової інформації, у якому визначається кількість QR-кодів за допомогою методу `byte_analysis()` (рядки 78–79) і відповідно до результату виконуються такі дії:

1. Якщо необхідний тільки один QR-код, то він формується за допомогою методу `generate_qr_code()` та додається до списку `image_list` (рядки 80–82).
2. Якщо необхідно два QR-коди, то текстова інформація ділиться нарівно, а для кожного з двох фрагментів викликається метод `generate_qr_code()`, щоб сформувати QR-коди (рядки 83–86). Потім обидва зображення додаються до списку `image_list` (рядок 87).
3. Якщо необхідно більше двох QR-кодів, то створюються додаткові змінні-лічильники (рядки 88–91), а проходження відбувається окремо по кожному символу тексту (рядок 92). Окремий символ перетворюється в бінарний формат (рядок 93), тоді відбуваються відповідні перевірки на можливість переповнення поточного QR-коду. Якщо поточний символ не переповнює QR-код, то змінюються значення відповідних лічильників із переходом до наступного символу (рядки 94–96). Якщо поточний символ повністю заповнить QR-код, то викликається метод `generate_qr_code()` для фрагмента тексту від попередньої точки переповнення (або 0 для першого QR-коду) до поточного символу включно (рядки 98–99). Цей QR-код додається до списку `image_list`, а значення змінних лічильників оновлюються (рядки 100–103). Якщо поточний символ переповнює QR-код, то викликається метод `generate_qr_code()` для фрагмента тексту від попередньої точки переповнення (або 0 для першого QR-коду) до попереднього символу (рядки 104–105). Отриманий QR-код додається до списку `image_list`, а значення змінних лічильників оновлюються (рядки 106–110). Якщо залишаються тільки два QR-коди, то решта тексту ділиться нарівно та створюються відповідні QR-коди кожного фрагмента з подальшим додаванням до списку `image_list` (рядки 111–116).

Остання категорія методів відповідає за декодування текстової інформації з QR-кодів. Так, використовуються методи `decode_qr_code()` для одиничного QR-коду та `decode_multi_qr_codes()` для списку QR-кодів у разі великої кількості текстової інформації. Реалізацію методів подано на рис. 3.10.

```python
    # генерація послідованості QR-кодів для великого об'єму текстих даних
    def generate_multi_qr_codes(self, in_text):
        bytes_num, qr_num = self.byte_analysis(in_text)
        image_list = []
        if qr_num == 1:
            tmp_image, _ = self.generate_qr_code(in_text)
            image_list.append(tmp_image)
        elif qr_num == 2:
            center_point = len(in_text) // 2
            tmp_image1, _ = self.generate_qr_code(in_text[:center_point])
            tmp_image2, _ = self.generate_qr_code(in_text[center_point:])
            image_list.extend([tmp_image1, tmp_image2])
        elif qr_num > 2:
            byte_counter = 0
            symbols_counter = 0
            prev_point = 0
            for symbol in in_text:
                num_bytes = len(bytes(symbol, 'utf-8'))
                if byte_counter + num_bytes < self.maxBytes:  # QR-код не заповнений повністю
                    symbols_counter += 1
                    byte_counter += num_bytes
                elif byte_counter + num_bytes == self.maxBytes:  # Qr-код заповнений
                    symbols_counter += 1
                    tmp_image, _ = self.generate_qr_code(in_text[prev_point:symbols_counter])
                    image_list.append(tmp_image)
                    prev_point = symbols_counter
                    byte_counter = 0
                    qr_num -= 1
                elif byte_counter + num_bytes > self.maxBytes:  # Поточний символ переповнить QR-код
                    tmp_image, _ = self.generate_qr_code(in_text[prev_point:symbols_counter])
                    image_list.append(tmp_image)
                    prev_point = symbols_counter
                    qr_num -= 1
                    symbols_counter += 1
                    byte_counter = num_bytes
                if qr_num == 2: # залишилося тільки 2 QR-кода
                    center_point = len(in_text[prev_point:]) // 2
                    tmp_image1, _ = self.generate_qr_code(in_text[prev_point:prev_point + center_point])
                    tmp_image2, _ = self.generate_qr_code(in_text[prev_point + center_point:])
                    image_list.extend([tmp_image1, tmp_image2])
                    break
        return image_list
```

Рис. 3.9. Фрагмент коду для реалізації методу `generate_multi_qr_codes()`

```
119     # декодування
120     @staticmethod
121     def decode_qr_code(qr_image):
122         # Декодування Qr-коду
123         data = pyzbar.decode(qr_image)
124         text_data = []
125         if data:
126             for qr_code in data:
127                 text_data = qr_code.data.decode('utf-8')
128         else:
129             text_data = "Error!"
130         return text_data
131 
132     # декодування множини QR-кодів в єдиний текст
133     def decode_multi_qr_codes(self, qr_image_list):
134         full_text = ""
135         # Декодування Qr-кодів
136         for qr_image in qr_image_list:
137             full_text += self.decode_qr_code(qr_image)
138         return full_text
```

Рис. 3.10. Фрагмент коду для реалізації методів декодування

Одиничне декодування зображення з QR-кодом виконується за допомогою методу `decode()` бібліотеки `pyzbar` (рядок 123). У разі успішного виконання інформація записується в змінну `text_data`, яка і є результатом роботи методу (рядки 124−130). Множинне декодування передбачає проходження по кожному зображенню QR-коду зі списку з подальшим викликом методу одиничного декодування (рядки 133−137). У результаті окремі фрагменти тексту поєднуються і повертаються як результат роботи методу (рядок 138).

3.3. Алгоритми спільного використання QR-кодів та LSB-стеганографії

Після генерації набору QR-кодів необхідно вирішити завдання їх ефективного вбудовування в зображення-обкладинку. Тому для підвищення ефективності роботи цього методу було враховано такі особливості вхідних і вихідних даних:

1. Зображення QR-кодів мають бути перетворені на формат у відтінках сірого (grayscale). Це дає змогу значно скоротити кількість пікселів зображення порівняно з кольоровою версією (втричі). Таке перетворення не має жодних похибок, оскільки QR-код є бінарною комбінацією чорних і білих фрагментів.
2. У звичайній LSB-стеганографії кожен піксель зображення подається в байтовому форматі з подальшою заміною бітів зображення-обкладинки на біти зображення-вкладення. Однак під час роботи з QR-кодами не потрібно зберігати кожен біт окремих пікселів. Натомість пропо-

нується кодувати в зображення-обкладинку лише 1 біт даних замість 8 (0 – для чорного кольору, 1 – для білого кольору). Такий підхід дає змогу значно збільшити потенційний обсяг зображення-обкладинки.

Клас для виконання стеганографії зображень отримав назву `SteganographyManager`. Необхідні бібліотеки для його функціонування та реалізацію конструктора класу подано на рис. 3.11.

```
import imutils
import numpy as np
from Modules.qr_codes import QrCodeManager
import cv2


class SteganographyManager:

    # конструктор
    def __init__(self):
        self.MASK_ZERO_1 = 0b11111110
        self.MASK_EXTRACT_1 = 0b00000001
        self.MASK_ZERO_2 = 0b11111100
        self.MASK_EXTRACT_2 = 0b00000011
        self.MASK_ZERO_4 = 0b11110000
        self.MASK_EXTRACT_4 = 0b00001111
        self.MASK_ZERO_01 = 0b11111101
        self.MASK_EXTRACT_01 = 0b00000010
        self.fixed_size = (370, 370)
        self.fixed_size_qr = (350, 350)
```

Рис. 3.11. Бібліотеки та конструктор класу `SteganographyManager`

На рис. 3.4 зверніть увагу на формування бінарних масивів даних і службової інформації. Бінарний масив службової інформації містить дані, необхідні в процесі декодування текстової інформації, закладеної в зображенні-обкладинці. Його обсяг становить 34 біти, у яких міститься така інформація: кількість пікселів даних на один QR-код – 24 біти; кількість закодованих QR-кодів – 8 бітів; метод кодування – 2 біти.

Бінарний масив даних є набором усіх бітів даних кожного з кодованих QR-кодів. Для його формування кожен QR-код перетворюється на одновимірний масив пікселів (рис. 3.12), і потім виконується аналіз значення яскравості кожного пікселя, у результаті якого кожному пікселю вихідного зображення в інформаційний масив додається 1 біт даних (0 − для чорного пікселя , 1 – для білого пікселя).

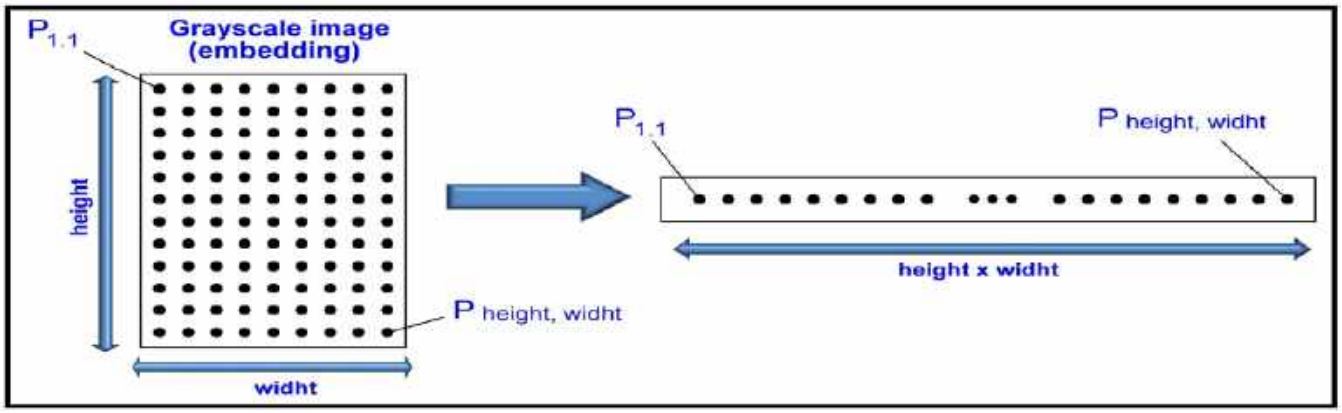


Рис. 3.12. Перетворення зображення у відтінках сірого на одновимірний масив

Основний метод класу `qr_steganography()` дає змогу кодувати масив зображень QR-кодів до зображення-обкладинки з дотриманням описаних раніше особливостей для максимізації потенційного обсягу обкладинки без ризику втрати корисної інформації в процесі. Алгоритм його роботи являє собою таку послідовність дій:

1. Зображення-обкладинка перетворюється на одновимірний масив пікселів, як показано на рис. 3.13.
2. У перші 34 пікселі додається інформація службового масиву відповідно до рис. 3.14.
3. Вбудовується масив даних у зображення-обкладинку.

Пропонується два методи вбудовування масиву даних. Перший передбачає заміну передостаннього біта пікселів зображення-обкладинки. Саме цим методом відбувається кодування службової інформації (див. рис. 3.14). Другий метод полягає в заміні одразу двох останніх бітів зображення-обкладинки (рис. 3.15) і дає змогу вдвічі скоротити кількість необхідних пікселів для вбудовування даних одного QR-коду в зображення-обкладинку. Недоліком другого підходу є його висока чутливість до шумів.

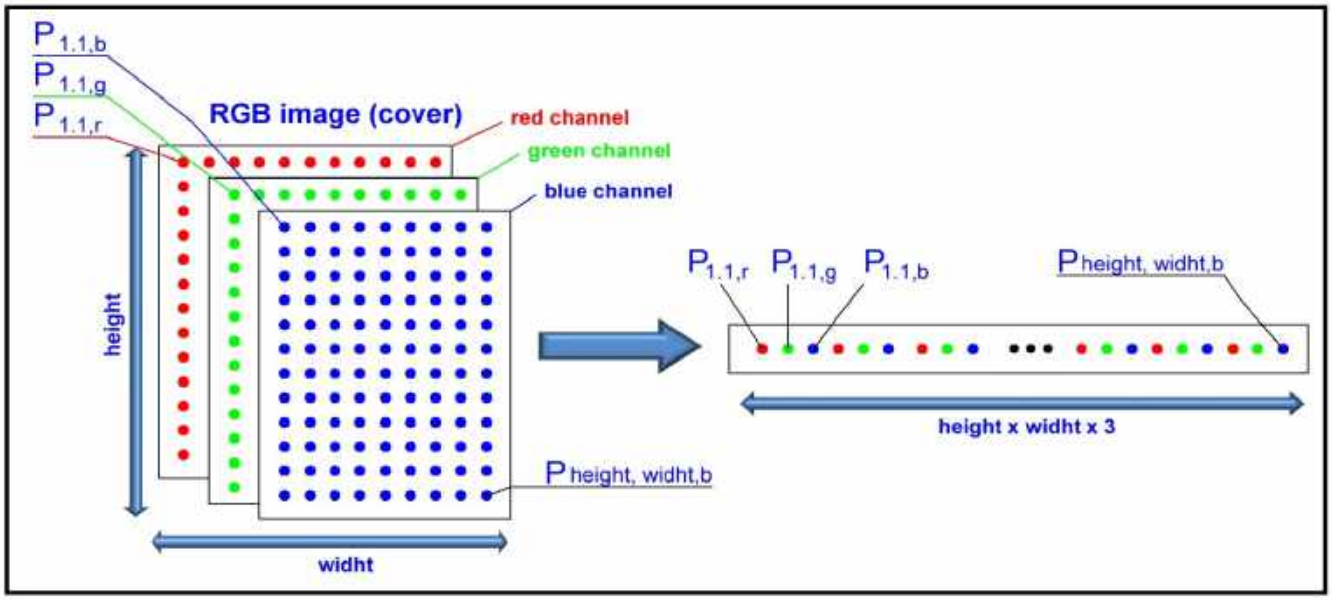


Рис. 3.13. Перетворення кольорового зображення на одновимірний масив

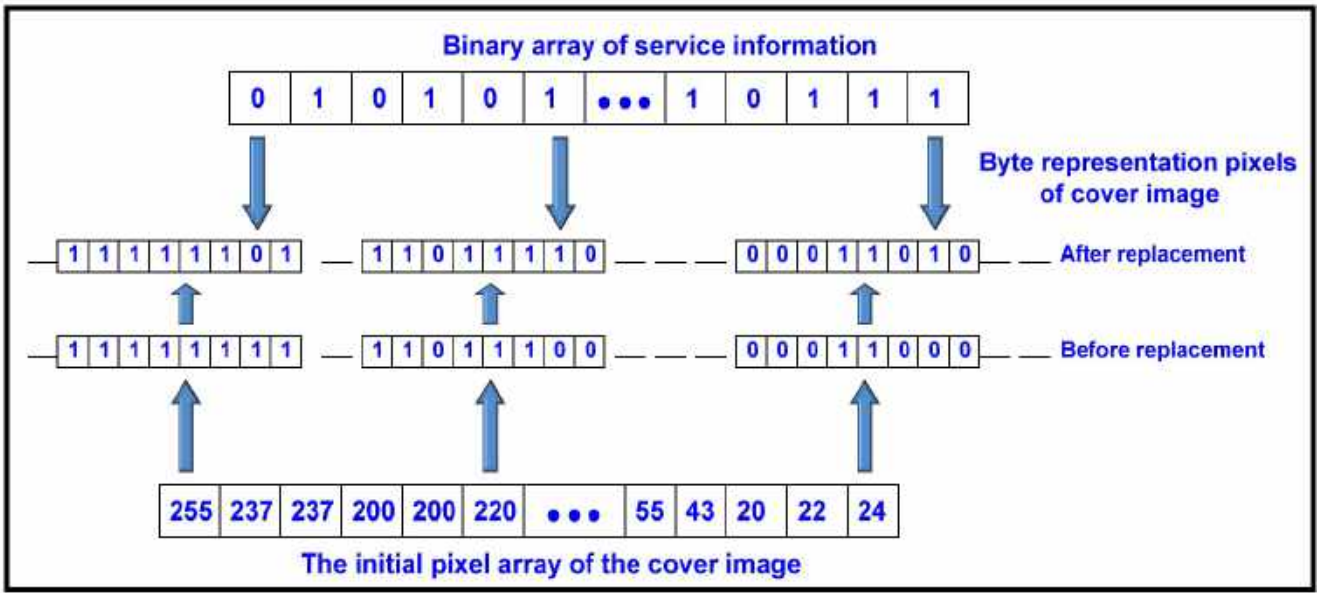


Рис. 3.14. Кодування службової інформації

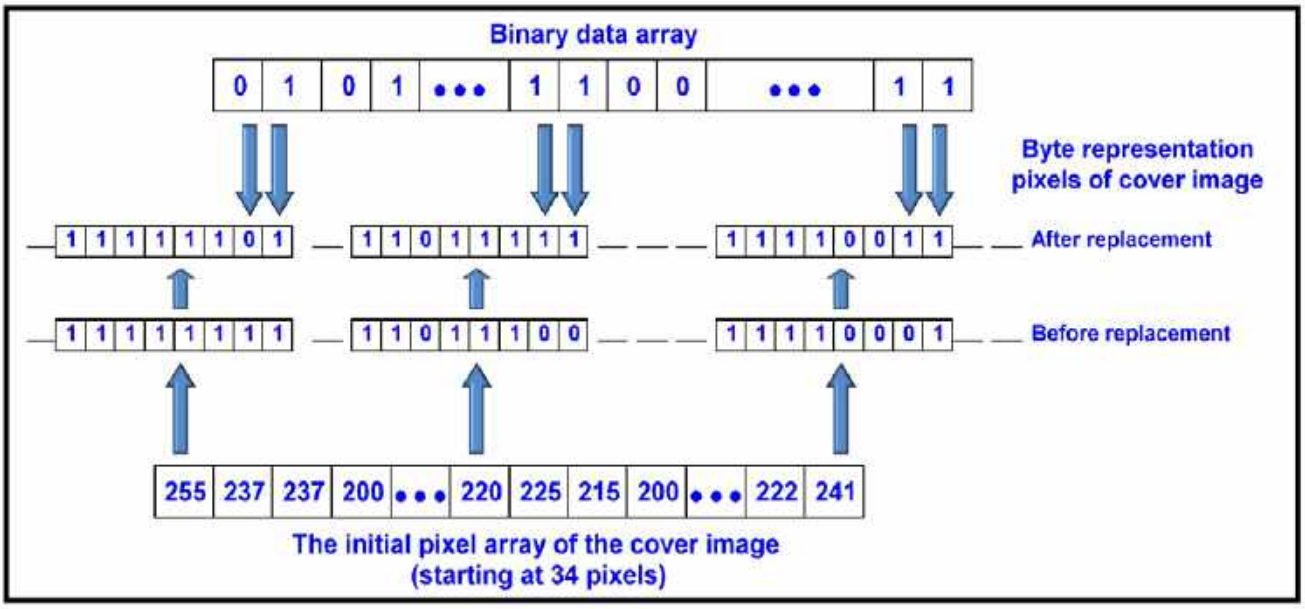


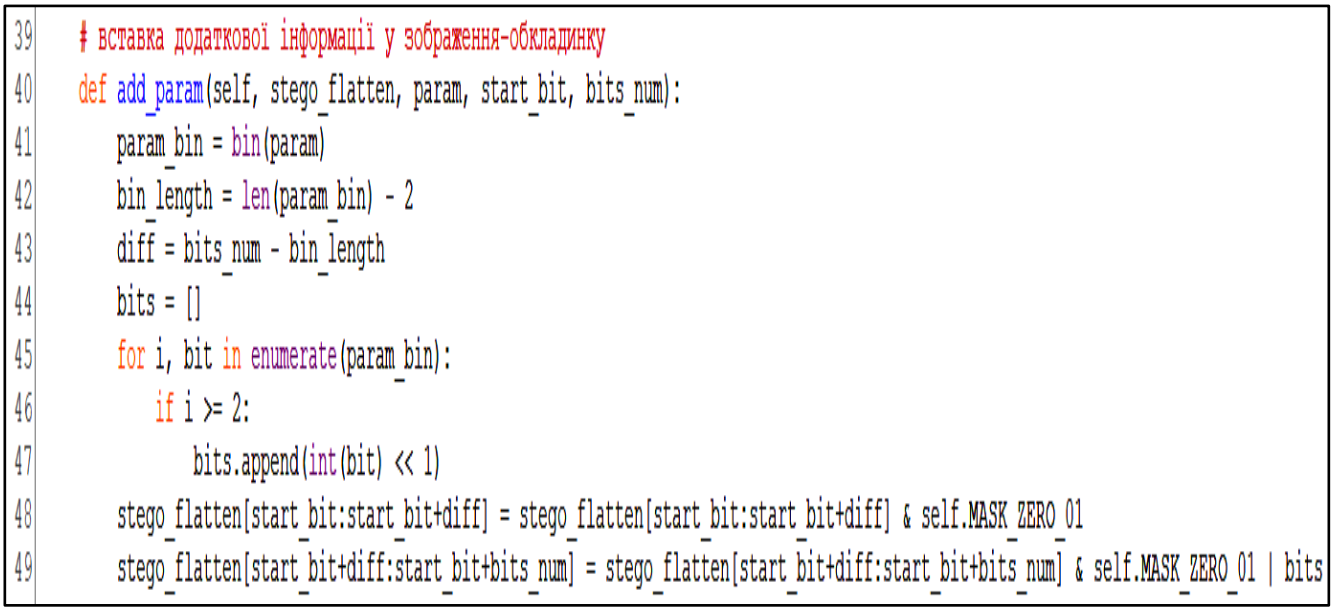
Рис. 3.15. Кодування основних даних

Завершальний етап вбудовування секретних даних передбачає зворотне перетворення масиву пікселів зображення-обкладинки на тривимірну матрицю. Так можна отримати перетворене зображення-обкладинку із закодованою інформацією всередині.

Реалізацію цього алгоритму в методі `qr_steganography()` подано на рис. 3.16. Також важливо зазначити, що в цьому методі використовуються два додаткові методи. Перший метод має назву `add_param()` та використовується для кодування службової інформації в зображення-обкладинку. Реалізацію цього методу наведено на рис. 3.17. Другий метод має назву `capacity_check()` та дає змогу виконувати перевірку ємності зображення-обкладинки відповідно до вибраного методу стеганографії та поточної кількості QR-кодів (рис. 3.18).

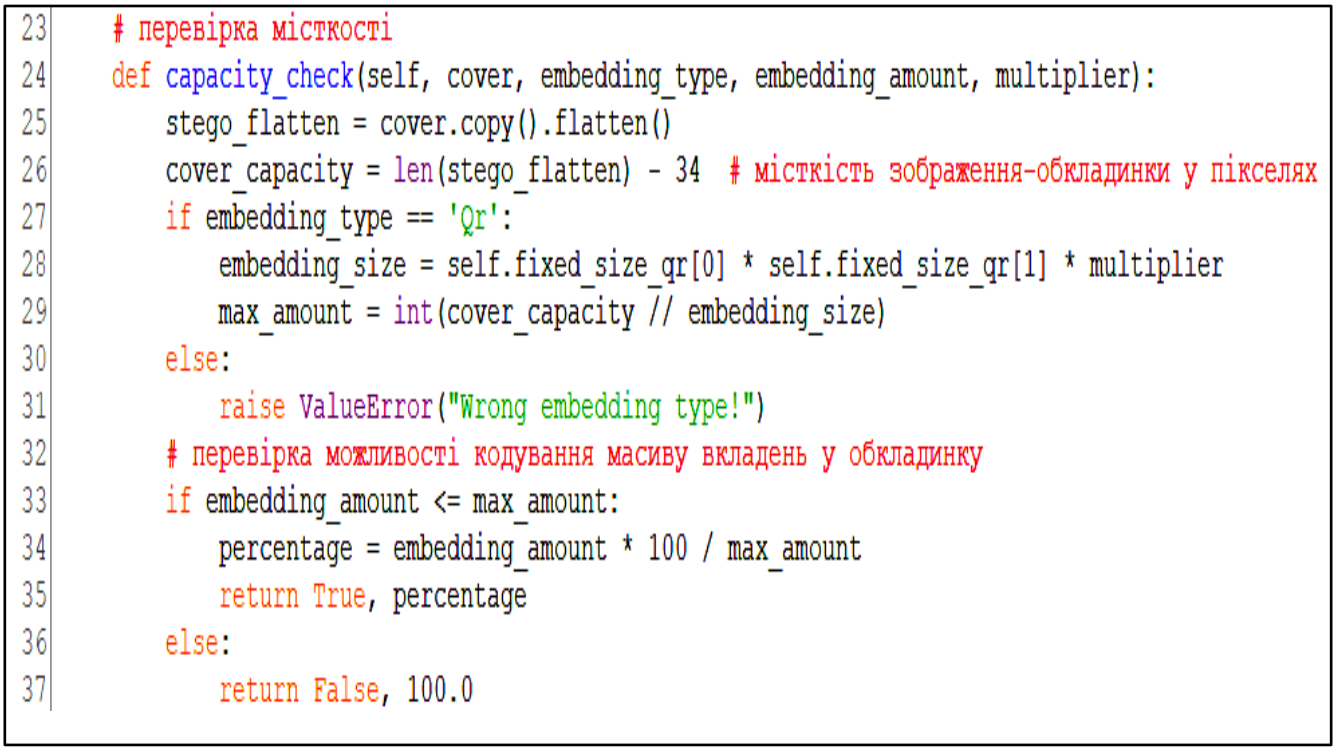
```python
    def qr_steganography(self, cover, embedding_massive, method):
        # визначення множника в залежності від обраного методу стеганографії
        if method == 0:
            multiplier = 1
        elif method == 1:
            multiplier = 0.5
        else:
            raise ValueError("Wrong method!")
        # перевірка можливості кодування масиву вкладень в обкладинку
        capacity_check, _ = self.capacity_check(cover, 'Qr', len(embedding_massive), multiplier)
        if capacity_check:
            stego_flatten = cover.copy().flatten()
            start_point = 34
            embedding_length = 0
            for index, embedding in enumerate(embedding_massive):
                embedding = cv2.resize(embedding, self.fixed_size_qr, cv2.INTER_AREA)
                # Формування біт для подальшого кодування
                values = []
                embedding_bits = []
                for pixel in embedding.flatten():
                    if 128 < pixel <= 255:
                        values.append(2)  # 0b0000 0010
                    elif 0 <= pixel <= 128:
                        values.append(0)  # 0b0000 0000
                    else:
                        raise ValueError('Wrong pixel value!')
            if method == 0:
                mask_zero = self.MASK_ZERO_01
                embedding_bits = np.array(values)
            elif method == 1:
                mask_zero = self.MASK_ZERO_2
                i = 0
                while i < len(values):
                    embedding_bits.append(values[i] | values[i+1] >> 1)
                    i += 2
                embedding_bits = np.array(embedding_bits)
            else:
                raise ValueError("Wrong method!")
            embedding_length = len(embedding_bits)
            # Приховування зображення в іншому зображенні
            end_point = start_point + embedding_length
            stego_flatten[start_point:end_point] = (stego_flatten[start_point:end_point] & mask_zero) | embedding_bits
            start_point = end_point
        # Кодування додаткової інформації
        self.add_param(stego_flatten, embedding_length, 0, 24)  # розмір вкладення
        self.add_param(stego_flatten, len(embedding_massive), 24, 8)  # кількість кодів
        self.add_param(stego_flatten, method, 32, 2)  # метод стеганографії
        return stego_flatten.reshape(cover.shape)
    else:
        raise ValueError("Exceeding the possible capacity of the container")  # Генерація помилки
```

Рис. 3.16. Фрагмент коду для реалізації методу `qr_steganography()`



```
39    # вставка додаткової інформації у зображення-обкладинку
40    def add_param(self, stego_flatten, param, start_bit, bits_num):
41        param_bin = bin(param)
42        bin_length = len(param_bin) - 2
43        diff = bits_num - bin_length
44        bits = []
45        for i, bit in enumerate(param_bin):
46            if i >= 2:
47                bits.append(int(bit) << 1)
48        stego_flatten[start_bit:start_bit+diff] = stego_flatten[start_bit:start_bit+diff] & self.MASK_ZERO_01
49        stego_flatten[start_bit+diff:start_bit+bits_num] = stego_flatten[start_bit+diff:start_bit+bits_num] & self.MASK_ZERO_01 | bits
```

Рис. 3.17. Фрагмент коду для реалізації методу `add_param()`



```
23    # перевірка місткості
24    def capacity_check(self, cover, embedding_type, embedding_amount, multiplier):
25        stego_flatten = cover.copy().flatten()
26        cover_capacity = len(stego_flatten) - 34  # місткість зображення-обкладинки у пікселях
27        if embedding_type == 'Qr':
28            embedding_size = self.fixed_size_qr[0] * self.fixed_size_qr[1] * multiplier
29            max_amount = int(cover_capacity // embedding_size)
30        else:
31            raise ValueError("Wrong embedding type!")
32        # перевірка можливості кодування масиву вкладень у обкладинку
33        if embedding_amount <= max_amount:
34            percentage = embedding_amount * 100 / max_amount
35            return True, percentage
36        else:
37            return False, 100.0
```

Рис. 3.18. Фрагмент коду для реалізації методу `capacity_check()`

Для декодування текстової інформації із зображення-обкладинки виконується зворотний прохід за алгоритмом, показаним на рис. 3.4. Із зображення-обкладинки виділяється і дешифрується службовий масив, а потім визначається обсяг і кількість закодованих QR-кодів. Після цього кожен QR-код відновлюється та додається до загального списку, який можна дешифрувати до єдиного тексту за допомогою методу `decode_multi_qr_codes()`, розглянутого в попередньому підрозділі.

У класі `SteganogaphyManager` алгоритм декодування реалізований у методі `qr_extract()` (рис. 3.19).

```
102     def qr_extract(self, stego_img):
103         stego_img = stego_img.flatten()
104         # Декодування параметрів стеганографії
105         qr_amount = 0
106         embedding_length = 0
107         method = 0
108         for i in range(34):
109             value = (stego_img[i] & self.MASK_EXTRACT_01) >> 1
110             if i < 24:
111                 embedding_length += pow(2, 23-i) * value
112             elif i < 32:
113                 qr_amount += pow(2, 31-i) * value
114             else:
115                 method += pow(2, 33 - i) * value
116         extracted_massive = []
117         start_point = 34
118         for num in range(qr_amount):
119             end_point = start_point + embedding_length
120             if method == 0:
121                 extracted_0 = []
122                 extracted_bytes_0 = stego_img[start_point:end_point] & self.MASK_EXTRACT_01
123                 for byte in extracted_bytes_0:
124                     if byte == 2:
125                         extracted_0.append(255)
126                     elif byte == 0:
127                         extracted_0.append(0)
128                 extracted_0 = np.array(extracted_0, dtype=np.uint8)
129                 extracted_0 = extracted_0.reshape(self.fixed_size_qr)
130                 extracted_massive.append(extracted_0)
```

```
131             if method == 1:
132                 extracted_1 = []
133                 extracted_bytes_1 = stego_img[start_point:end_point] & self.MASK_EXTRACT_2
134                 for byte in extracted_bytes_1:
135                     if byte == 3:
136                         extracted_1.append(255)
137                         extracted_1.append(255)
138                     elif byte == 2:
139                         extracted_1.append(255)
140                         extracted_1.append(0)
141                     elif byte == 1:
142                         extracted_1.append(0)
143                         extracted_1.append(255)
144                     elif byte == 0:
145                         extracted_1.append(0)
146                         extracted_1.append(0)
147                 extracted_1 = np.array(extracted_1, dtype=np.uint8)
148                 extracted_1 = extracted_1.reshape(self.fixed_size_qr)
149                 extracted_massive.append(extracted_1)
150             start_point = end_point
151             if len(extracted_massive) == qr_amount:
152                 break
153         return extracted_massive
```

Рис. 3.19. Друга частина фрагменту коду для реалізації методу `qr_extract()`

4. ОЦІНЮВАННЯ ЕФЕКТИВНОСТІ ЗАХОДІВ ЩОДО ЗАХИСТУ ІНФОРМАЦІЇ

Ще під час постановки завдання проведення досліджень зазначалося, що проблема підвищення якості роботи систем оптичного розпізнавання тексту має комплексний характер. Можна виділити чотири основні напрями досліджень, результати яких дадуть можливість суттєво покращити основні робочі й експлуатаційні характеристики проєктованих і наявних систем:

1. Необхідно далі вдосконалювати методи виявлення та сегментації тексту − дві найбільш важливі та складні дослідні завдання в галузі комп'ютерного зору.
2. Дуже важливо ефективно компенсувати зовнішні впливи найбільш значущих негативних факторів у процесі підготовки фотознімків текстових документів (геометричні спотворення зображень, затінені фрагменти знімків, низька контрастність, шуми тощо).
3. Слід створити набір зручних сервісних функцій (перегляд даних на будь-якому етапі перетворень, сегментація та анонімізація окремих фрагментів тексту, виділення рисунків, друк і збереження результатів роботи в стандартних форматах).
4. Передбачити комплексні заходи щодо захисту конфіденційної інформації, отриманої після розпізнавання текстових документів, під час передавання її незахищеними каналами зв'язку. Слід спроєктувати захист інформації так, щоб вона була універсальною та давала змогу працювати з будь-якою системою розпізнавання текстів.

Зауважимо, що розробленню найважливішого комплексу заходів щодо захисту інформації під час передавання конфіденційних даних відкритими каналами зв'язку наразі приділяється недостатньо уваги з боку дизайнерів сучасних систем оптичного розпізнавання текстів. Також майже немає універсальної методики оцінювання ефективності інформаційного захисту. У цьому розділі буде детально описано підхід до розв’язання цієї проблеми.

Ефективність роботи систем оптичного розпізнавання текстів оцінюється комплексно з урахуванням якості розпізнавання символів, ємності інформаційних носіїв стеганографії та показників скритності переданих даних у сеансах зв'язку. Якість роботи OCR оцінювалося теоретично й експериментально. Розглянемо ці питання детальніше.

4.1. Визначення ємності каналу зв'язку

Для оцінювання ефективності кодування текстової інформації введемо показник інформаційної ємності стеганографії (information capacity), $\mathrm{I_c}$. Показник являє собою інформаційну завантаженість одного пікселя зображення-обкладинки:

$$\mathrm{I_c} = \frac{\mathrm{N_{bytes}}}{\mathrm{N_{pixels}}}, \tag{4.1}$$

де N_{bytes} − кількість байтів закодованого в зображення-обкладинку тексту; N_{pixels} – кількість пікселів зображення-обкладинки.

Зазначимо, що цей показник залежить не тільки від вибраного методу кодування (у нашому випадку за допомогою QR-кодів), а й від розміру зображення-обкладинки. Це пов'язано з тим, що для кодування QR-коду в зображенні-обкладинці має бути певна кількість пікселів. Але завжди залишаються невикористані пікселі, які впливатимуть на зміну максимально можливої інформаційної ємності цього зображення-обкладинки. Що стосується розмірів зображень QR-кодів, то під час експериментальних досліджень було встановлено їх оптимальне значення 350x350 пікселів. Використання такого розміру дає змогу усунути помилки під час декодування QR-коду, які виникають у разі подальшого зменшення його розмірів.

Обчислимо значення інформаційної ємності за двох методів кодування для зображень-обкладинок стандартних розмірів. Нагадаємо, що перший метод це заміна тільки передостаннього біта пікселя зображення-обкладинки на біт пікселя зображення, що маскується, а другий метод − заміна відразу двох молодших бітів пікселів обкладинки на біти пікселя зображення, що маскується. Результати розрахунків занесено до табл. 4.1.

Таблиця 4.1

Оцінювання інформаційної ємності різних зображень-обкладинок

Розмір зображення-обкладинки	Перший метод		Другий метод	
	Ic^{max}, байт/піксель	Залишок пікселів	Ic^{max}, байт/піксель	Залишок пікселів
600 x 400	$2.04 \cdot 10^{-2}$	107 466	$4.48 \cdot 10^{-2}$	46 216
800 x 600	$2.24 \cdot 10^{-2}$	92 466	$4.68 \cdot 10^{-2}$	31 216
1024 x 768	$2.36 \cdot 10^{-2}$	31 762	$4.72 \cdot 10^{-2}$	31762
1280 x 1024	$2.38 \cdot 10^{-2}$	12 126	$4.77 \cdot 10^{-2}$	12 126
1600 x 900	$2.37 \cdot 10^{-2}$	32 466	$4.75 \cdot 10^{-2}$	32 466
1920 x 1080	$2.35 \cdot 10^{-2}$	95 766	$4.76 \cdot 10^{-2}$	34 516

Очевидно, що найкращі показники інформаційної ємності можна отримати, використовуючи зображення-обкладинки розмірів 1024x768. Це пояснюється невеликою кількістю пікселів, що не використовуються. Значення цих показників для розмірів зображень-обкладинок 1280x1024, 1600x900, 1920x1080 також цілком прийнятні. Найгірші результати були отримані для зображення-обкладинки з розмірами 600x400. Це поясню-

ється занадто великою кількістю пікселів, що не використовуються, за умови порівняно невеликих розмірів обкладинки.

Також можна помітити значне збільшення показника під час використання другого методу кодування, що пояснюється зменшенням фактичної кількості пікселів, необхідної для приховання окремого QR-коду. При цьому важливо зазначити, що в табл. 4.1 наведено максимально можливі значення для кожного варіанта розмірів зображення-обкладинки. Цього можна досягти лише за наявності достатньої кількості QR-кодів. Отже, для максимізації значень інформаційної ємності в робочих умовах необхідно добирати контейнер найменшого розміру, який може вмістити необхідну кількість QR-кодів.

4.2. Аналіз якості стеганографічного зображення

Вимірювання якості зображення – складний і трудомісткий процес, оскільки на думку людини впливають фізичні та психологічні параметри. Для вимірювання якості зображення пропонується безліч методів, але жоден із них не вважається ідеальним для вимірювання якості. Оцінка якості зображення відіграє роль у галузі оброблення зображень. Було проведено безліч досліджень щодо вимірювання якості зображення. Хороші показники якості зображень IQ (Image Quality) мають бути точними та послідовними в прогнозуванні якості. Більшість показників якості зображень пов'язані з різницею між двома зображеннями (вихідним і спотвореним зображенням).

Зазначимо, що якість зображення погіршується під час зберігання, оброблення, стиснення, передавання каналами зв'язку тощо, тому це потребує об'єктивного кількісного оцінювання зміни якості.

Під час оціннювання якості зображення зазвичай використовуються такі два методи: суб'єктивний та об'єктивний. Суб'єктивне оціннювання методу вважається дорогим і трудомістким, оскільки треба вибрати кілька спостерігачів, потім показати їм кілька зображень і попросити кожного особисто оцінити якість зображень. Середня оцінка думок – добре відомий підхід до суб'єктивного оцінювання якості зображення. У цьому підході групу людей просять порівняти вихідні зображення зі спотвореними зображеннями, щоб оцінити якість спотвореного зображення. Середній бал приймається як якість зображення. Незважаючи на те, що цей процес відображає людське сприйняття, він вважається трудомістким і непрактичним для використання в поєднанні з іншими алгоритмами оброблення зображень. Тому потрібна об'єктивна метрика, щоб корелювати із суб'єктивним оцінюванням.

Об'єктивне оцінювання використовує автоматичні алгоритми оцінювання якості зображення без втручання людини. Матриці об'єктивної якості залежать від наявності вихідного зображення:

- повністю доступне еталонне зображення;
- частковий доступ до еталонної інформації;

- немає доступу до еталонного зображення, це також називається «сліпим оцінюванням якості».

Далі зосередимося на розгляді повної еталонної метрики об'єктивної якості. Маємо як вихідні, так і спотворені стеганографічні зображення. Тому орієнтуватимемося на повноцінні показники якості. Заходи якості повного еталонного зображення можна поділити на шість класів об'єктивного оцінювання зображення, а саме:

1) на основі різниці пікселів (середньоквадратична помилка (MSE), відношення сигнал/шум (SNR) та пікове відношення сигнал/шум (PSNR) − ці показники легко оцінити);

2) на основі кореляції (кореляція використовується для вимірювання різниці між двома цифровими зображеннями, під час оцінювання якості зображення кореляція пікселів використовується як міра якості зображення);

3) на основі меж (у цьому класі розміщуються межі у вихідному та спотвореному зображеннях, потім використовується міра зміщення положення меж або їх узгодженість для визначення якості зображення для всього зображення);

4) на основі спектральної відстані (до вихідного та спотвореного зображень застосовується дискретне перетворення Фур'є; різниця амплітуди Фур'є, або фазового спектру, використовується як міра якості зображення);

5) на основі контексту (замість порівняння пікселів у вихідних і спотворених зображеннях околи пікселів порівнюються один з одним шляхом визначення ймовірності багатовимірного контексту для вимірювання якості зображення);

6) на основі зорової системи людини (HVS) (якість зображення вимірюється так само як людським оком; люди зазвичай використовують зміни контрасту, кольору та частоти у своїх вимірах).

Дослідження області повного еталонного HVS пов'язані з розумінням зорового сприйняття людини, де якість обчислюється шляхом порівняння його з еталонним зображенням. Перші п'ять типів метрик якості зображення є так званими метриками помилок простої статистики, а останній називається метрикою, що ґрунтується на нелінійних функціях.

Останнім часом було зроблено багато зусиль для розроблення адекватних та об'єктивних показників якості зображення. Показники якості MSE, PSNR, NCC, SSIM та UIQI (їх визначення будуть детально описані далі) є об'єктивними показниками якості зображення, що найчастіше використовуються.

Наведемо найбільш поширені заходи, які використовують для порівняння вбудовуваних стегозображень S та зображень-обкладинок C розміром MxN:

1. ***Peak-signal-to-noise ratio*** (відношення пікового сигналу до шуму). PSNR визначається як статистична оцінка якості зображення, що

використовується для вимірювання спотворення між обкладинкою та вкладеним зображенням. Великі значення PSNR означають невелику кількість спотворень і приводять до високої якості зображення, тоди як малі значення − до помітних змін стеганографічних зображень, що легко виявляються за допомогою HVS (зорової системи людини). PSNR надається формулою

$$\mathrm{PSNR} = 10\log_{10}\frac{255^2}{\mathrm{MSE}}, \tag{4.2}$$

де середньоквадратична помилка MSE визначається виразом

$$\mathrm{MSE} = \frac{1}{\mathrm{M}\times\mathrm{N}}\sum_{\mathrm{i}=0}^{\mathrm{M}-1}\sum_{\mathrm{j}=0}^{\mathrm{N}-1}(\mathrm{S}-\mathrm{C})^2\,. \tag{4.3}$$

2. ***Normalized-cross-correlation*** (нормалізована крос-кореляція). NCC показує, наскільки сильно стеганографічне зображення корелює з обкладинкою. Значення NCC лежить у діапазоні від 0 до 1. Якщо значення NCC дорівнюють 1, це означає, що стегозображення повністю стійкі до різних атак оброблення зображень. NCC можна подати у вигляді формули

$$\mathrm{NCC} = \sum_{\mathrm{i}=0}^{\mathrm{M}-1}\sum_{\mathrm{j}=0}^{\mathrm{N}-1}\big(\mathrm{S}(\mathrm{i},\mathrm{j})\times\mathrm{C}(\mathrm{i},\mathrm{j})\big) \Big/ \sum_{\mathrm{i}=0}^{\mathrm{M}-1}\sum_{\mathrm{j}=0}^{\mathrm{N}-1}\big(\mathrm{S}(\mathrm{i},\mathrm{j})\big)^2\,. \tag{4.4}$$

3. ***Structural similarity index measurement*** (вимірювання індексу структурної подібності). SSIM – метрика оцінювання якості зображення, яка порівнює два зображення (стего й обкладинки) для отримання подібності між ними. Пропонується як покращення PSNR. SSIM набуває форми

$$\mathrm{SSIM}(\mathrm{C},\mathrm{S}) = \frac{(2\mu_\mathrm{C}\mu_\mathrm{S}+\mathrm{c}_1)\times(2\sigma_\mathrm{CS}+\mathrm{c}_2)}{(\mu_\mathrm{C}^2+\ \mu_\mathrm{S}^2+\mathrm{c}_1)\times(\sigma_\mathrm{C}^2\sigma_\mathrm{S}^2+\mathrm{c}_2)}, \tag{4.5}$$

де μ_C − середнє значення C пікселів обкладинки; μ_S − середнє значення S пікселів стегозображення; σ_C^2 – дисперсія C; σ_S^2 − дисперсія S, σ_CS − коваріація C і S; $\mathrm{c}_1 = (\mathrm{K}_1\mathrm{L})^2$ та $\mathrm{c}_2 = (\mathrm{K}_2\mathrm{L})^2$ − дві змінні, що стабілізують поділ зі слабким знаменником; L − динамічний діапазон значень пікселів, $\mathrm{K}_1 = 0{,}01$ і $\mathrm{K}_2 = 0{,}03$ за замовчуванням.

4. ***Universal image quality index*** (універсальний індекс якості зображення). UIQI використовується для оцінювання візуальної якості зображень. Великі значення UIQI означають, що вкладене зображення та зображення-обкладинки сильно кореловані, а відмінності між ними дуже малі. Універсальний індекс якості UIQI можна розрахувати за допомогою формули

$$\mathrm{UIQI} = \frac{4\sigma_{\mathrm{x,y}}\bar{\mathrm{x}}\bar{\mathrm{y}}}{\big(\sigma_\mathrm{x}^2+\sigma_\mathrm{y}^2\big)[(\bar{\mathrm{x}})^2+(\bar{\mathrm{y}})^2]}\,, \tag{4.6}$$

де

$$\bar{x} = \frac{1}{M \times N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (x_{ij}) \ , \bar{y} = \frac{1}{M \times N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (y_{ij}), \tag{4.7}$$

а дисперсії окремих компонентів обчислюються за допомогою формул

$$\sigma_x^2 = \frac{1}{(M \times N) - 1} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (x_{ij} - \bar{x})^2 , \tag{4.8}$$

$$\sigma_y^2 = \frac{1}{(M \times N) - 1} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (y_{ij} - \bar{y})^2 , \tag{4.9}$$

$$\sigma_{xy}^2 = \frac{1}{(M \times N) - 1} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (x_{ij} - \bar{x})(y_{ij} - \bar{y}) , \tag{4.10}$$

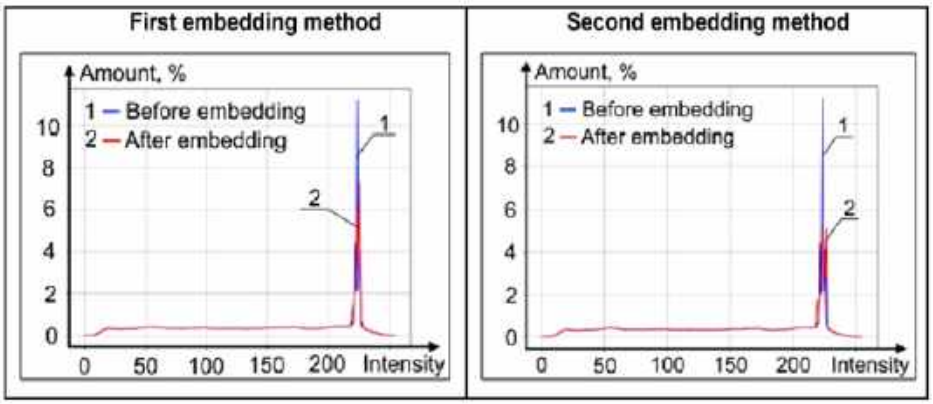
де x − значення пікселів зображення обкладинки; y − значення пікселів стегозображення; $\bar{x}$ − середнє значення x; $\bar{y}$ − середнє значення y; σ_x^2, σ_y^2 та σ_{xy}^2 − дисперсія та коваріація зображень **x** та **y** відповідно.

4.3. Результати експериментальних досліджень

Оскільки найбільш важливим показником якості роботи системи захисту є скритність факту передавання даних, були проведені експериментальні дослідження з оцінювання змін зображення-обкладинки в результаті вбудовування в неї секретних зображень. Для цього було створено бібліотеку зображень-обкладинок із різною розмірністю та різними умовами освітлення сцени (світлі та темні знімки). Набір цих зображень із зазначенням їх розмірності показано на рис. 4.1.

Рис. 4.1. Бібліотека зображень-обкладинок для тестування якості стеганографічних перетворень

Для проведення досліджень використовувалися візуальні й об'єктивні характеристики зображення-обкладинки до та після процедури вкладення в неї зображень QR-кодів із конфіденційним текстом одним із двох запропонованих методів LSB-стеганографії. На думку авторів, найбільш наочною візуальною характеристикою властивостей і виду зображення-обкладинки до та після стеганографічних перетворень є нормована гістограма розподілу яскравостей. Для зручності зіставлення гістограми до і після перетворення показані різними кольорами в одному вікні шляхом прямого накладання. Ці показники наведено на рис. 4.2. Їх аналіз показав, що запропоновані процедури вкладення QR-кодів із зашифрованим текстом помітно не спотворюють зображення-обкладинку незалежно від її розмірів та розподілу яскравостей у полі зображення. Ці спотворення становлять частки відсотка загальної кількості пікселів цієї яскравості. Тому можна впевнено вважати, що ці зміни виявляться візуально непомітними за умови використання як першого, так і другого методів заміни молодших бітів (одного або двох бітів) у байті для кожного пікселя. Зауважимо, що відхилення двох гістограм зображення-обкладинки мають хаотичний шумовий характер.



а

б

Рис. 4.2. Гістограми зображень-обкладинок до та після стеганографії: *а* − гістограми для зображення-обкладинки А; *б* − гістограми для зображення-обкладинки В; *в* − гістограми для зображення-обкладинки С; *г* − гістограми для зображення-обкладинки Д; *д* − гістограми для зображення-обкладинки Е; *е* − гістограми для зображення-обкладинки F

в

г

д

Рис. 4.2. Продовження

е

Рис. 4.2. Закінчення

Кількісний аналіз похибок, що виникають у результаті стеганографічних перетворень, проводився статистичними методами. Досліджувалась залежність показника PSNR від кількості зображень QR-кодів максимальної інформаційної ємності вкладених у зображення-обкладинку відомих розмірів. Результати цих досліджень в узагальненому вигляді показано на рис. 4.3 і 4.4.

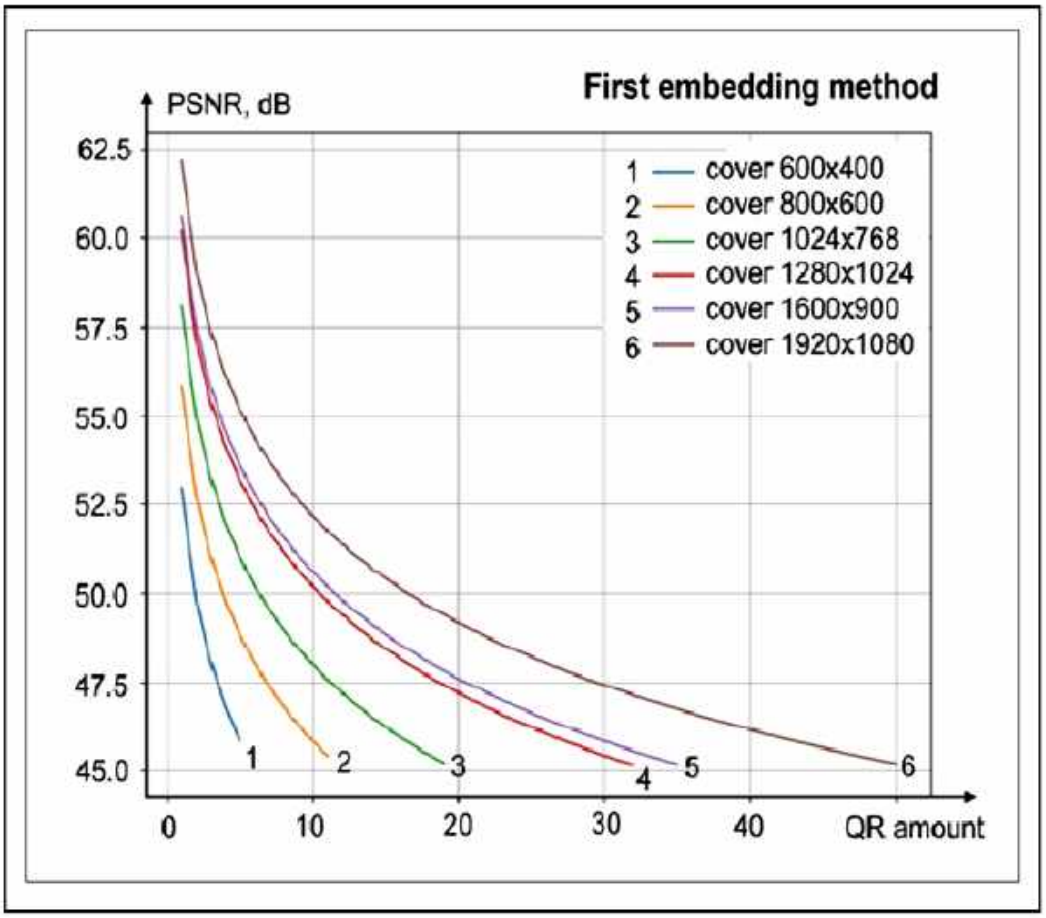


Рис. 4.3. Залежність показника PSNR від кількості вбудованих QR-кодів першим методом (заміною молодшого біта)

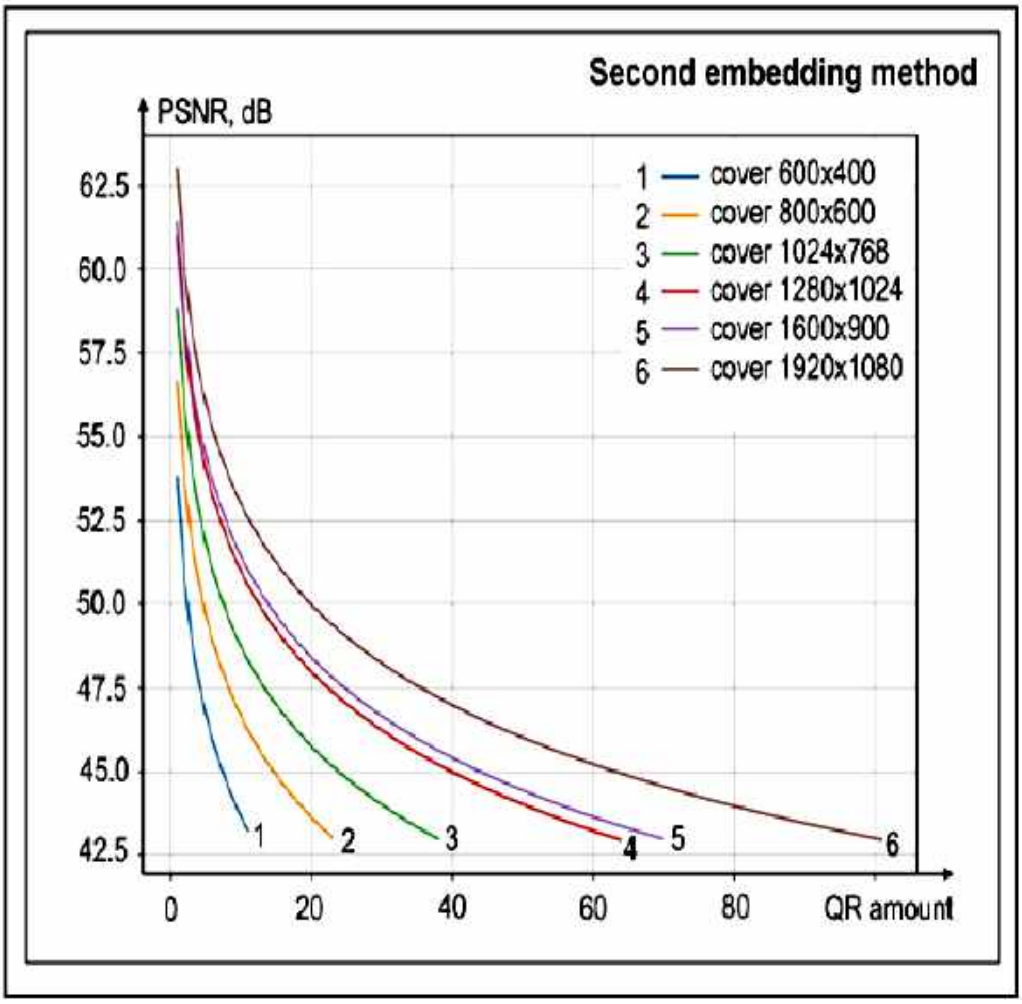


Рис. 4.4. Залежність показника PSNR від кількості вбудованих QR-кодів другим методом (заміною двох молодших бітів)

На наведених вище рисунках подано два варіанти розрахунку показника PSNR для першого та другого варіантів заміни молодших бітів у числах, що характеризують яскравість кожного пікселя пропонованими методами LSB-стеганографії. Зрозуміло, що найсприятливіші (великі – на рівні 55...60 dB) значення PSNR характерні для невеликої кількості вкладених QR-кодів, оскільки площа, яку займає вкладене зображення, є мінімальною. Показник PSNR експоненційно зменшується в міру збільшення кількості QR-кодів, що впроваджуються, отже, знижується якість маскування фактів передавання секретних повідомлень. Найбільш сприятливий варіант – зображення-обкладинка великої розмірності та порівняно невелика кількость QR-кодів.

Далі досліджувався ступінь кореляції стеганографічного зображення із зображенням-обкладинкою за значеннями показника NCC. На рис. 4.5 і 4.6 унаочнена залежність показників нормалізованої крос-кореляції NCC від кількості вкладених у зображення-контейнер QR-кодів. Зміни цього показника досліджувалися для двох методів вкладення молодших бітів у байт, що характеризує яскравість пікселя зображення-обкладинки. Для повноти аналізу ці зображення-обкладинки також вибиралися різних розмірів.

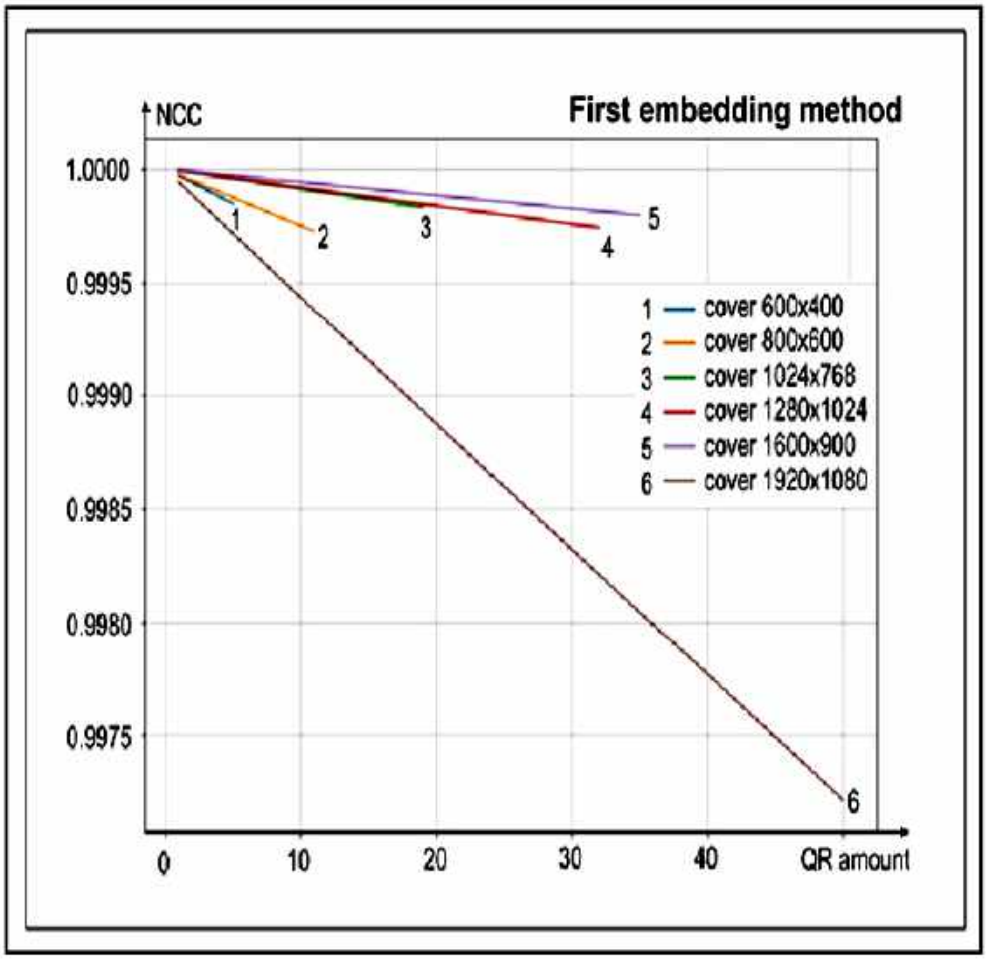


Рис. 4.5. Аналіз показника крос-колеляції NCC від кількості вкладених QR-кодів першим методом (заміною молодшого біта)

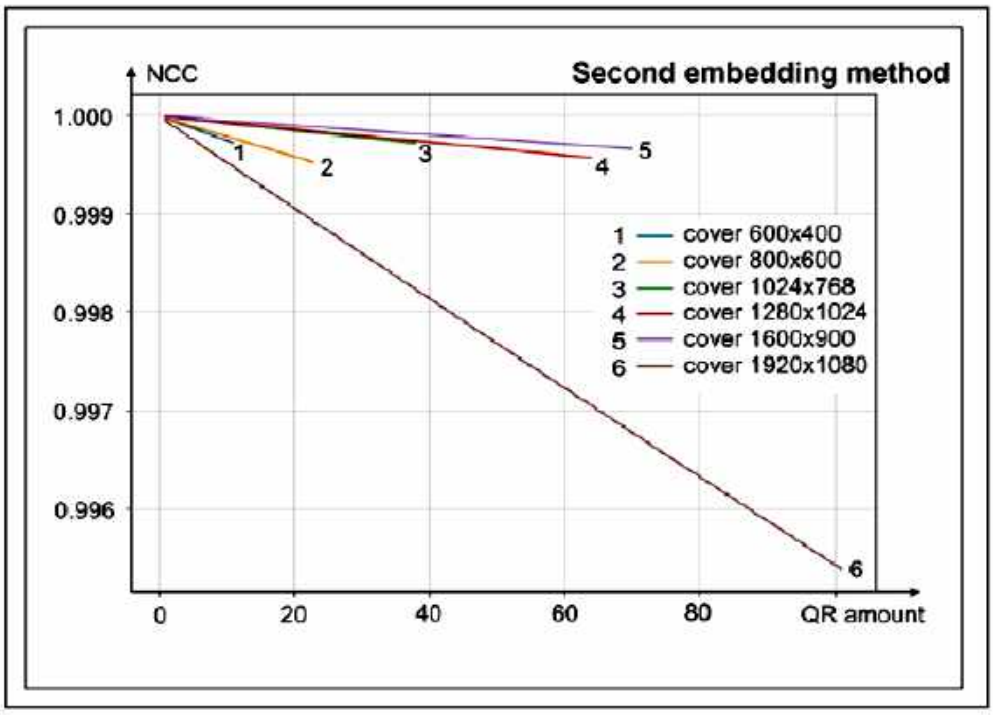


Рис. 4.6. Аналіз показника крос-колеляції NCC від кількості вкладених QR-кодів другим методом (заміною двох молодших бітів)

Для обох методів кодування стеганографічних зображень зображення-контейнер кореляція між ними залишається дуже високою (NCC дорівнює ~ 0,9999) і слабо залежить від збільшення кількості вбудованих QR-кодів. Зменшення NCC має лінійний характер. Високий ступінь крос-кореляції між стегозображенням та зображенням-обкладинкою забезпечує стійкість до різних атак оброблення зображень і свідчить про правильний підхід до вибору методів кодування даних.

На наступному етапі досліджень оцінювалась реальна мінливість індексу структурної подібності стегозображення та зображення-обкладинки SSIM залежно від кількості вкладених QR-кодів. Результати розрахунків наведено на рис. 4.7 та 4.8. Детальний аналіз показав, що для всіх співвідношень розмірів стегозображень (кількості QR-кодів) та зображень-обкладинок нормований індекс SSIM наближений до 1 і незначно знижується зі збільшенням кількості вкладених QR-кодів. Це характерно відразу для двох методів LSB-стеганографії (заміні лише молодшого біта або відразу двох молодших бітів зображення-обкладинки на біти стегозображення). І це також є свідченням правильного вибору методів кодування конфіденційної текстової інформації для її передавання каналами зв'язку.

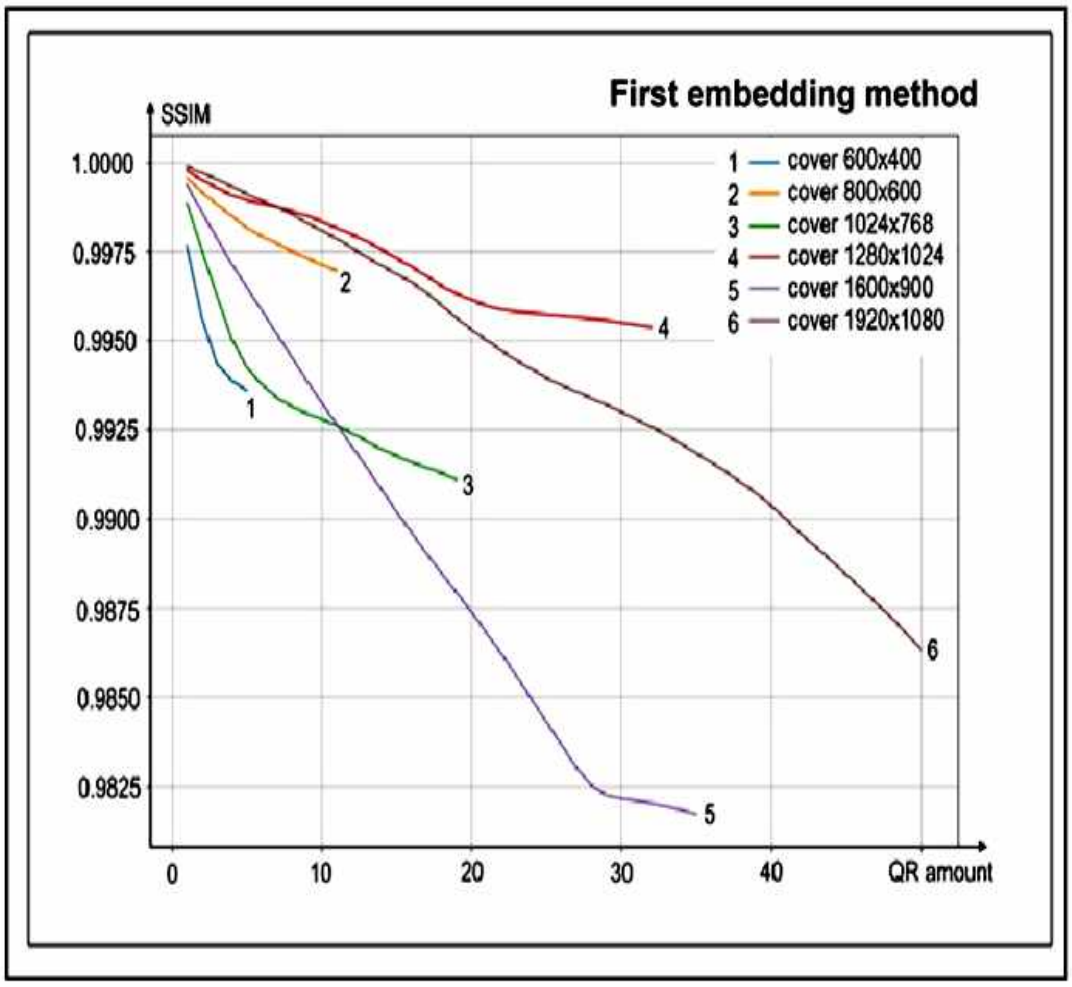


Рис. 4.7. Залежність індексу структурної подібності SSIM від кількості вкладених QR-кодів першим методом (заміною молодшого біта)

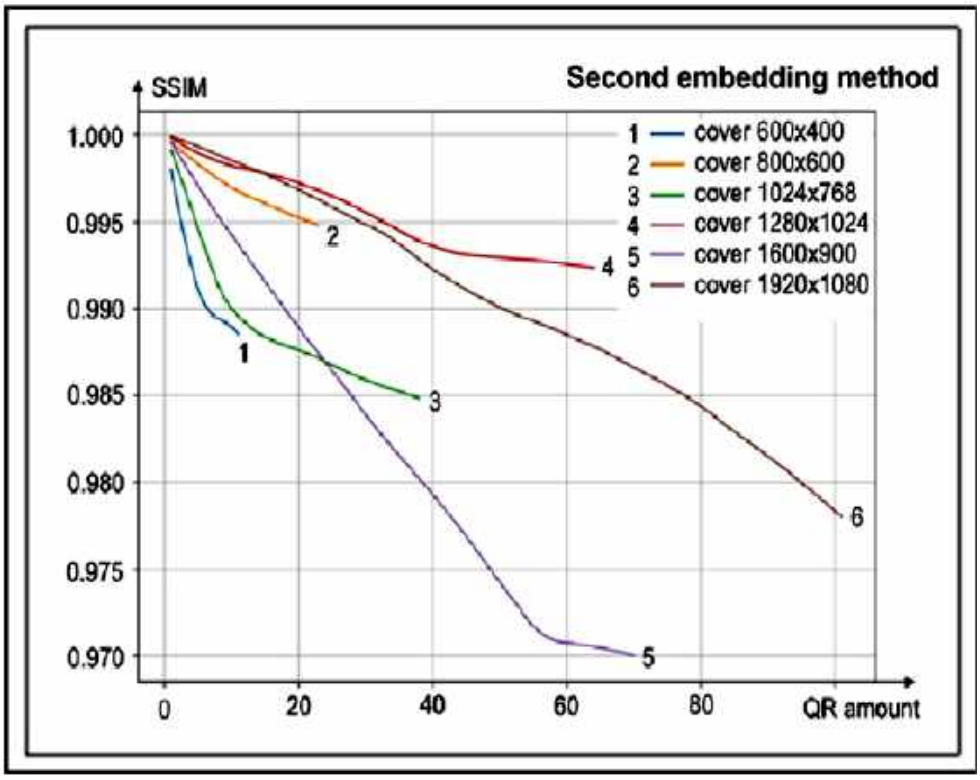


Рис. 4.8. Залежність індексу структурної подібності SSIM від кількості вкладених QR-кодів другим методом (заміною двох молодших бітів)

Завершальним етапом досліджень якості зображення-обкладинки став аналіз UIQI (універсального індексу якості зображення). Результати аналізу показано на рис. 4.9 та 4.10.

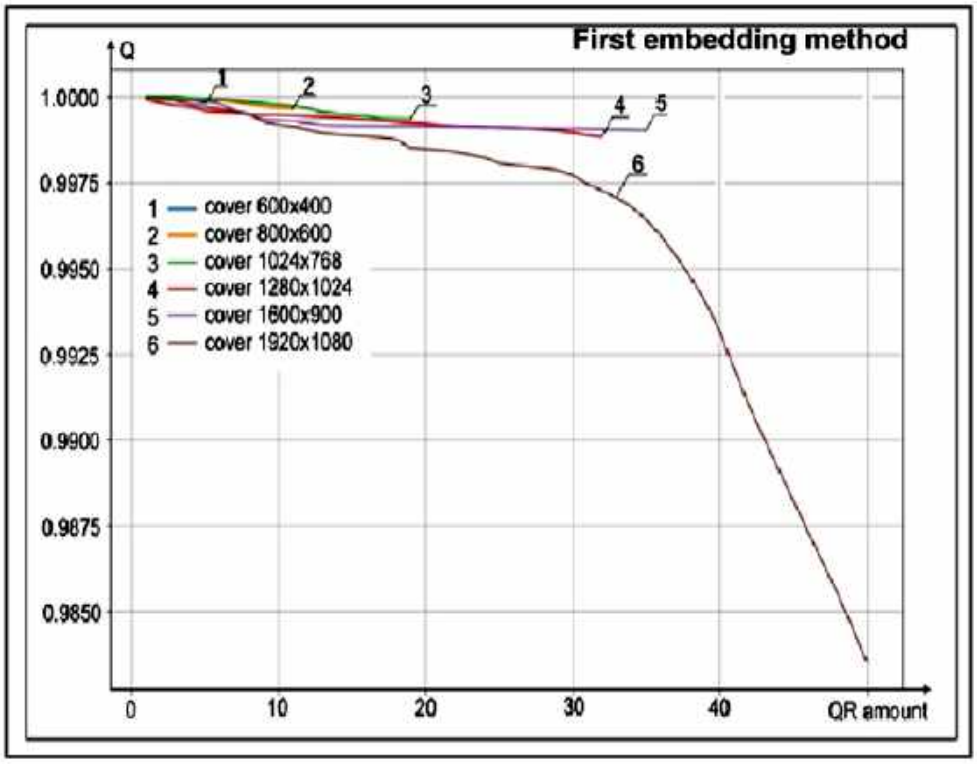


Рис. 4.9. Залежність універсального індексу якості зображення Q від кількості вкладених QR-кодів першим методом (заміною молодшого біта)

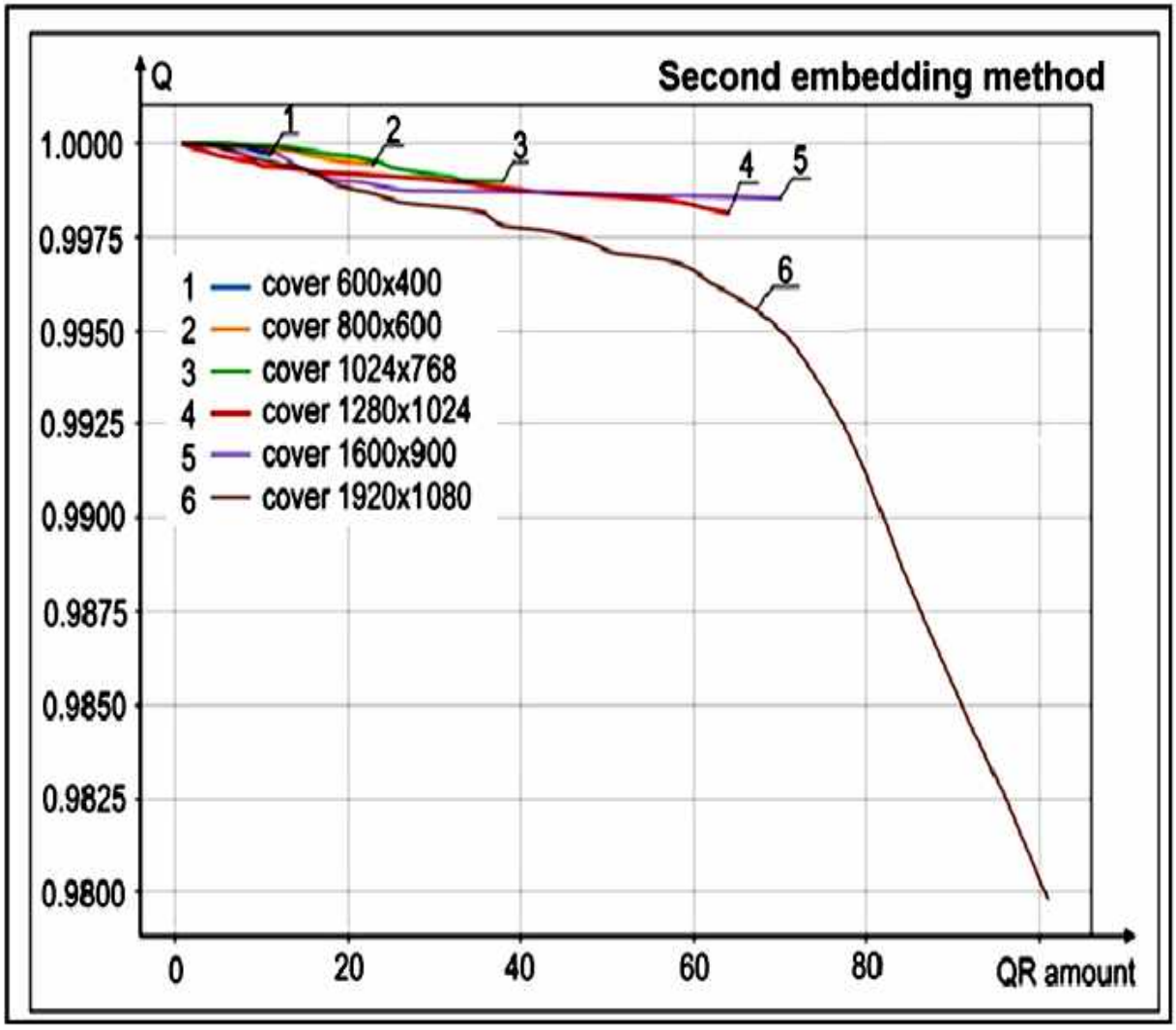


Рис. 4.10. Залежність універсального індексу якості зображення Q від кількості вкладених QR-кодів другим методом (заміною двох молодших бітів)

Можна помітити, що, як і в попередніх дослідженнях, нормовані значення універсального індексу Q високі і наближаються до 1 за умови всіх співвідношень QR-кодів, що вкладаються, до розмірів зображення-обкладинки. Це також є явною позитивною ознакою ефективного спільного використання QR-кодів та алгоритмів LSB-стеганографії.

Таким чином, у цьому розділі наведено опис експериментальних досліджень, метою яких було об'єктивне оцінювання якості кодування даних системи оптичного розпізнавання тексту для їх передавання відкритими каналами в мережі «Інтернет». Оцінювалася ефективність використання QR-кодування текстів у поєднанні з алгоритмами LSB-стеганографії. Розгорнутий статистичний аналіз і використання ефективних візуальних маркерів (нормованих гістограм розподілу яскравості зображення-обкладинки до та після процедури стеганографії) доводить високу ефективність запропонованих методів захисту інформації. Розроблена методика може бути використана в проєктах схожої спрямованості.

5. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ OCR

5.1. Структура системи OCR, основні модулі, класи та методи

Побудова сучасної високопродуктивної системи оптичного розпізнавання тексту, орієнтованої на її захист від негативного впливу збурювальних факторів, передбачає чітку послідовність оброблення вихідних даних, а також оптимізацію структури процедури розпізнавання. Як зазначалося раніше, робота систем розпізнавання текстової документації зазвичай не потребує оброблення даних у реальному часі. Проте часто потрібні інтерактивні процедури (такі як перегляд, сегментація тексту тощо), які значно уповільнюють роботу програми. Ці особливості визначають необхідний функціонал і структуру програми оптичного розпізнавання тексту. Так, загальну діаграму класів розробленої програми оптичного розпізнавання тексту показано на рис. 5.1.

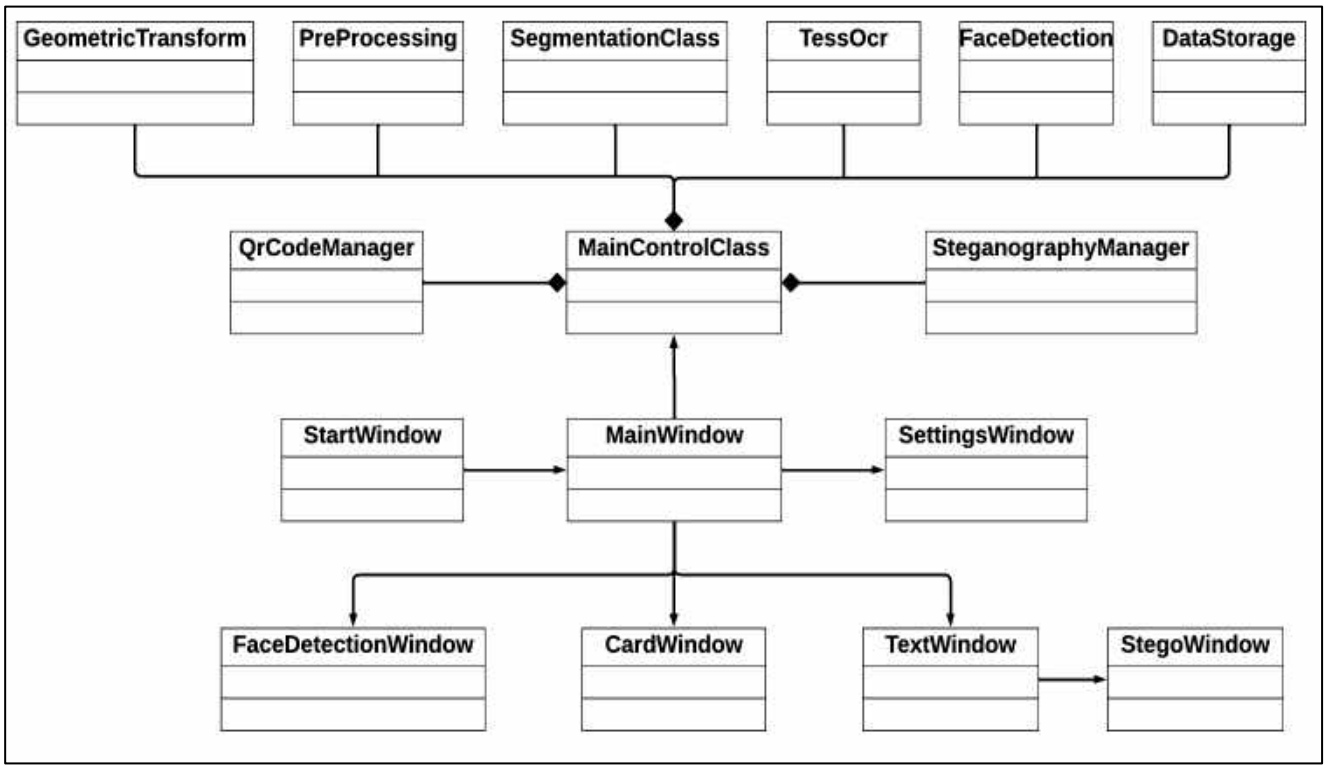


Рис. 5.1. Діаграма класів програми оптичного розпізнавання тексту

Як можна побачити, окрім описаних у попередніх розділах класів для геометричних перетворень, попереднього оброблення зображення, сегментації та анонімізації, розпізнавання тексту, роботи з QR-кодами та стеганографії, були додані такі класи:

- `FaceDetection` для реалізації можливості детектування облич на зображенні;
- `DataStorage` для забезпечення роботи з каталогами даних на пристрої, тобто їх завантаження, видалення, перейменування, збереження результатів розпізнавання у форматах `*docx, *pdf`;

- `MainControlClass` як клас-інтерфейс, який поєднує в собі роботу всіх основних класів програми та дає змогу ефективно виконувати окремі функціональні можливості програми;
- `StartWindow`, `MainWindow`, `TextWindow`, `CardWindow`, `FaceDetectionWindow`, `SettingsWindow`, `StegoWindow` як класи інтерфейсу для окремих вікон програми.

Далі розглянемо структуру класів `FaceDetection`, `DataStorage` та `MainControlClass` із зазначенням методів, атрибутів і загального принципу їх роботи.

Клас `FaceDetection` на основі OpenCV був спроєктований для вирішення базового завдання виділення облич на зображенні. Атрибути та методи класу подані в табл. 5.1.

Таблиця 5.1

Атрибути та методи класу `FaceDetection`

Найменування	Призначення
Атрибути	
`_original_image`	Початкове зображення для подальшого виділення облич. Тип даних – тривимірний масив (`array`)
`face_cascade` `eye_cascade`	Класифікатори для виділення облич та очей на зображенні. Тип даних – каскадні класифікатори (`OpenCV CascadeClassifier`)
Методи	
`__init__`	Конструктор класу. Які параметри приймає: • `image` – початкове зображення для подальшого оброблення
`face_detection`	Метод для виділення облич на зображенні. Не приймає параметрів. Результат роботи: • `image` – зображення із виділеними обличчями; • `image_with_eyes` – зображення із виділеними обличчями й очима; • `roi_massive` – список виділених облич

Використання цього класу дає змогу здійснювати виділення облич та очей на зображенні зі збереженням отриманих результатів, які можуть бути використані сторонніми ресурсами для подальшої ідентифікації осіб.

Завдання оптичного розпізнавання тексту пов'язано з використанням великої кількості зображень із текстом, для яких доречно використовувати окремі каталоги даних. Такі каталоги можуть містити як самі зображення для оброблення, так і результати розпізнавання у форматах `pdf`, `doc`, `txt` тощо.

Для автоматизації роботи з такими каталогами був спроєктований окремий клас `DataStorage`, який дає змогу запам'ятовувати поточний каталог, зчитувати наявну в ньому інформацію з можливістю її оновлення, видаляти або перейменовувати файли та записувати нові файли, отримані в результаті роботи системи оптичного розпізнавання тексту. Атрибути та методи класу наведено в табл. 5.2.

Таблиця 5.2

Атрибути та методи класу `DataStorage`

Найменування	Призначення
Атрибути	
`_path`	Шлях до каталогу даних. Тип даних – рядок (`string`)
`_required_catalogs`	Список імен, необхідних для роботи каталогів. Тип даних – список (`list`)
`_pictures_croping_path`	Шлях для збереження рисунків. Тип даних – рядок (`string`)
`_face_detection_results_path`	Шлях для збереження результатів виділення облич. Тип даних – рядок (`string`)
`_temp_path`	Шлях для збереження тимчасових файлів. Тип даних – рядок (`string`)
`_containers_path`	Шлях для збереження зображень-обкладинок. Тип даних – рядок (`string`)
`_initial_path`	Поточний шлях, при якому викликаються операції з даними. Тип даних – рядок (`string`)
`_all_data_names`	Список з іменами всіх даних у каталозі. Тип даних – список (`list`)
`_image_names`	Список з іменами всіх зображень у каталозі. Тип даних – список (`list`)

Продовження табл. 5.2

Найменування	Призначення
Атрибути	
`_doc_file_names`	Список з іменами всіх doc-файлів у каталозі. Тип даних – список (`list`)
`_png_file_names`	Список з іменами всіх `png`-файлів у каталозі. Тип даних – список (`list`)
`_txt_file_names`	Список з іменами всіх `txt`-файлів у каталозі. Тип даних – список (`list`)
Методи	
`__init__`	Конструктор класу. Які параметри приймає: • `path` – шлях до каталогу даних
`_data_update`	Метод для оновлення поточної інформації. Не приймає параметрів
`_to_path`	Метод для переходу до каталогу з даними. Які параметри приймає: • `path_name` – найменування каталогу для переходу
`_return_to_initial_path`	Метод для повернення до початкового шляху. Не приймає параметрів
`catalog_check`	Метод для перевірки наявності необхідних для роботи каталогів. Не приймає параметрів
`data_from_path`	Метод для зчитування інформації з каталогу та заповнення відповідних інформаційних масивів. Не приймає параметрів
`delete_file`	Метод для видалення файлу з каталогу. Які параметри приймає: • `filename` – ім’я файлу для видалення
`rename_file`	Метод для перейменування файлу з каталогу. Які параметрив приймає: • `filename` – ім’я файлу для перейменування; • `new_name` – нове ім’я файлу

Продовження табл. 5.2

Найменування	Призначення
Методи	
open_file	Метод для відкриття файлу. Які параметри приймає: • filename – ім'я файлу для відкриття
read_file	Метод для зчитування зображення з каталогу. Які параметри приймає: • filename – ім'я файлу для зчитування; • mode – режим зчитування. Результат роботи: • result – зчитане зображення.
get_container	Метод для зчитування зображення-обкладинки. Які параметри приймає: • filename – ім'я файлу зображення. Результат роботи: • cover – зчитане зображення-обкладинка; • cover_pixmap – зображення-обкладинка у форматі pixmap
save_temp	Метод для збереження тимчасової інформації. Які параметри приймає: • filename – ім'я файлу для збереження; • image – тимчасове зображення для збереження
save_face_detection_results	Метод для збереження результатів виділення облич. Які параметри приймає: • roi_massive – список зображень із виділеними обличчями
save_croped_pictures	Метод для збереження окремо виділених рисунків. Які параметри приймає: • pictures – список виділених рисунків
doc_write	Метод для запису файлу у форматі doc до каталогу. Які параметри приймає: • filename – ім'я нового файлу; • text – текст для збереження

Закінчення табл. 5.2

Найменування	Призначення
Методи	
`png_write`	Метод для запису файлу у форматі png до каталогу. Які параметри приймає: • `filename` – ім'я нового файлу; • `image` – зображення для збереження
`txt_write`	Метод для запису файлу у форматі txt до каталогу. Які параметри приймає: • `filename` – ім'я нового файлу; • `text` – текст для збереження
`txt_expand`	Метод для запису нової інформації до наявного файлу у форматі txt. Які параметри приймає: • `filename` – ім'я наявного файлу; • `text` – текст для збереження

`get_data_names`
`get_image_names`
`get_doc_file_names`
`get_png_file_names`
`get_txt_file_names`
`get_path`
	Методи для отримання поточної інформації.
	Не приймають параметрів.
	Результат роботи:
	• Повернення копій відповідних атрибутів класу

Спроєктований клас `DataStorage` дає змогу значною мірою прискорити роботу із каталогами даних, надаючи необхідний і зручний функціонал для зчитування, модифікації та запису інформації.

Останнім для ознайомлення є клас `MainControlClass`, який керує роботою всіх розглянутих раніше класів для забезпечення зручного інтерфейсу виконання конкретних операцій без необхідності використання окремих класів. Таким чином, ознайомившись із методами й атрибутами цього класу, користувач може виконувати такі операції, як встановлення поточного каталогу даних, виконання геометричних перетворень, попереднього оброблення, сегментації та анонімізації зображення, розпізнавання тексту та його кодування засобами стеганографії. Усю роботу з окремими класами приховано в реалізації методів і налагоджено для виконання конкретної мети. Атрибути та методи класу подано в табл. 5.3.

Таблиця 5.3

Атрибути та методи класу `MainControlClass`

Найменування	Призначення
Атрибути	
`_dataStorage`	Екземпляр класу `DataStorage` для роботи з каталогами даних. Тип даних – `DataStorage`
`original_image`	Початкове зображення з текстом. Тип даних – тривимірний масив (`array`)
`geometric_image`	Зображення після виконання геометричного перетворення. Тип даних – тривимірний масив (`array`)
`filtering_image`	Зображення після виконання фільтрації шумів. Тип даних – двовимірний масив (`array`)
`hf_filtering_image`	Зображення після виконання високочастотної фільтрації. Тип даних – двовимірний масив (`array`)
`binarization_image`	Зображення після виконання бінаризації. Тип даних – двовимірний масив (`array`)
`erosion_image`	Зображення після виконання ерозії. Тип даних – двовимірний масив (`array`)
`final_image`	Кінцеве зображення перед розпізнаванням тексту. Тип даних – двовимірний масив (`array`)
`roi_massive`	Список виділених сегментів тексту. Тип даних – список (`list`)
`sa_manager`	Екземпляр класу `SegmentationClass` для виконання операцій сегментації та анонімізації. Тип даних – `SegmentationClass`
Методи	
`__init__`	Конструктор класу. Не приймає параметрів

Продовження табл. 5.3

Найменування	Призначення
Методи	
`set_initial_data`	Метод для встановлення поточного каталогу та зчитування необхідних даних. Які параметри приймає: • `full_name` – повний шлях до файлу зображення з каталогу даних. Результат роботи: • `flag` – логічне значення успішності виконання методу; • `filename` – ім'я файлу зображення без шляху
`geometric_preparations`	Метод для виконання етапу геометричних перетворень і збереження отриманого зображення як тимчасового файлу. Які параметри приймає: • `filename` – ім'я початкового зображення в каталозі; • `combo_box_name` – ім'я тимчасового файлу для збереження
`manual_geometric_transform`	Метод для виконання етапу геометричних перетворень інтерактивним методом і збереження отриманого зображення як тимчасового файлу. Які параметри приймає: • `filename` – ім'я початкового зображення в каталозі; • `combo_box_name` – ім'я тимчасового файлу для збереження
`image_preparations`	Метод для виконання етапу попереднього оброблення та збереження отриманих зображень як тимчасових файлів. Які параметри приймає: • `settings` – список настройок для виконання всіх операцій попереднього оброблення зображення; • `operation_flags` – список логічних значень для перевірки необхідності виконання кожної з операцій; • `combo_box_names` – список імен тимчасових файлів для збереження

Продовження табл. 5.3

Найменування	Призначення
Методи	
segmentation	Метод для виконання сегментації та збереження отриманого зображення як тимчасового файлу. Які параметри приймає: • combo_box_name – ім'я тимчасового файлу для збереження
anonymization	Метод для виконання анонімізації та збереження отриманого зображення як тимчасового файлу. Які параметри приймає: • combo_box_name – ім'я тимчасового файлу для збереження
clear_sa_results	Метод для очищення результатів сегментації та анонімізації. Не приймає параметрів
picture_croping	Метод для виділення та збереження окремих рисунків. Не приймає параметрів
face_detection	Метод для виділення облич на зображенні з подальшим збереженням результатів. Результат роботи: • roi_massive – список зображень із виділеними обличчями
qr_code_generation	Метод для створення QR-коду за текстом і збереження отриманого зображення як тимчасового файлу. Які параметри приймає: • text – текст для кодування. Результат роботи: • qr_image – зображення QR-коду; • flag – логічне значення успішності виконання методу
ocr	Метод для виконання оптичного розпізнавання тексту. Які параметри приймає: • language – мова тексту для розпізнавання. Результат роботи: • text – розпізнана текстова інформація

Закінчення табл. 5.3

Найменування	Призначення
Методи	
`steganography_encoding`	Метод для створення QR-коду за текстом із подальшим виконанням стеганографії. Які параметри приймає: • `text` – текст для кодування; • `cover` – зображення-обкладинка; • `method` – метод кодування. Результат роботи: • `stego` – зображення-обкладинка із закодованим текстом
`steganography_decoding`	Метод для декодування текстової інформації з зображення-обкладинки. Які параметри приймає: • `filename` – ім'я файлу зображення-обкладинки із закодованим текстом. Результат роботи: • `text` – декодована текстова інформація
`calculate_occupancy`	Метод для аналізу наповненості зображення-обкладинки під час стеганографії. Які параметри приймає: • `text` – текст для кодування; • `cover` – зображення-обкладинка; • `method` – метод кодування. Результат роботи: • `capacity_check` – логічне значення успішності виконання перевірки; • `percentage_value` – значення наповненості у відсотках

5.2. Програмний інтерфейс

Проєктування зрозумілого та компактного інтерфейсу є однією з головних цілей під час розроблення системи оптичного розпізнавання тексту, оскільки дає змогу значною мірою спростити використання всього запропонованого системою функціонала для користувача.

Для проєктування інтерфейсних вікон програми застосовувався `QtDesigner` – вільне середовище для розроблення графічних інтерфейсів (GUI) програм, що використовують бібліотеку `Qt`. `Qt Designer` дає змогу створювати графічні інтерфейси користувача за допомогою низки інструментів (рис. 5.2). Існує панель інструментів «Панель віджетів», у якій дос-

тупні для використання елементи інтерфейсу – віджети. Кожен віджет має свій набір властивостей, що визначається відповідним класом бібліотеки Qt.

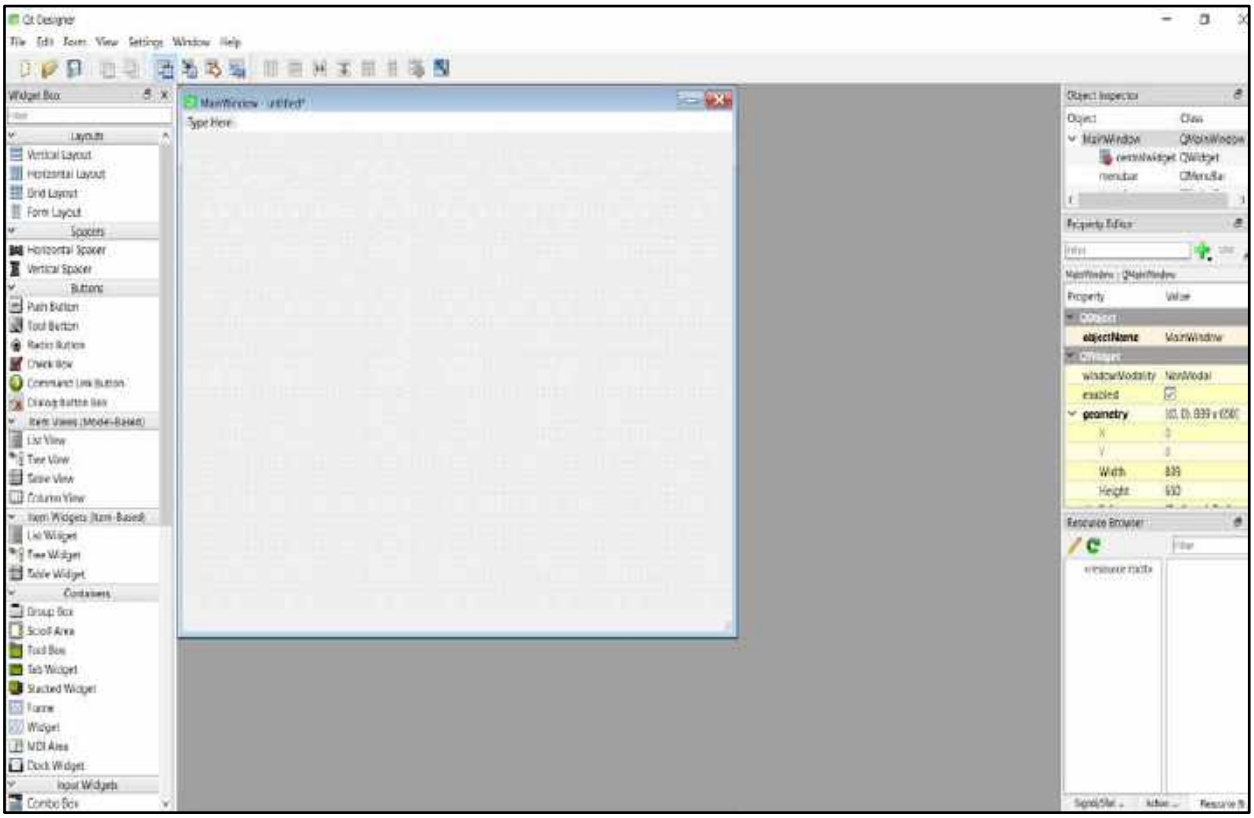


Рис. 5.2. Інтерфейс QtDesigner

Розроблений інтерфейс зберігається у файл із розширенням ui, який підключається до створюваної програми за допомогою спеціальних методів бібліотеки Qt. Цей файл має xml-формат і може в разі потреби редагуватись у будь-якому текстовому редакторі.

Під час побудови програмного інтерфейсу було поставлено завдання проєктування макета головного вікна програми, у якому користувач повинен мати доступ до всіх функціональних можливостей системи оптичного розпізнавання тексту. Відповідно до цього було поставлено такі цілі для його проєктування:

1) підвікно перегляду поточних файлів у каталозі даних;
2) кнопки «Delete» та «Rename» для роботи з файлами;
3) можливість вибору режиму сортування файлів у підвікні перегляду за алфавітом або типом даних;
4) два пусті поля фіксованих розмірів для відображення зображень на різних етапах попереднього оброблення;
5) випадні списки для обох зображень із можливістю вибору етапу попереднього оброблення.

Отриманий у результаті макет інтерфейсу головного вікна з виділенням елементів, відповідних за окремі цілі проєктування, наведено на рис. 5.3.

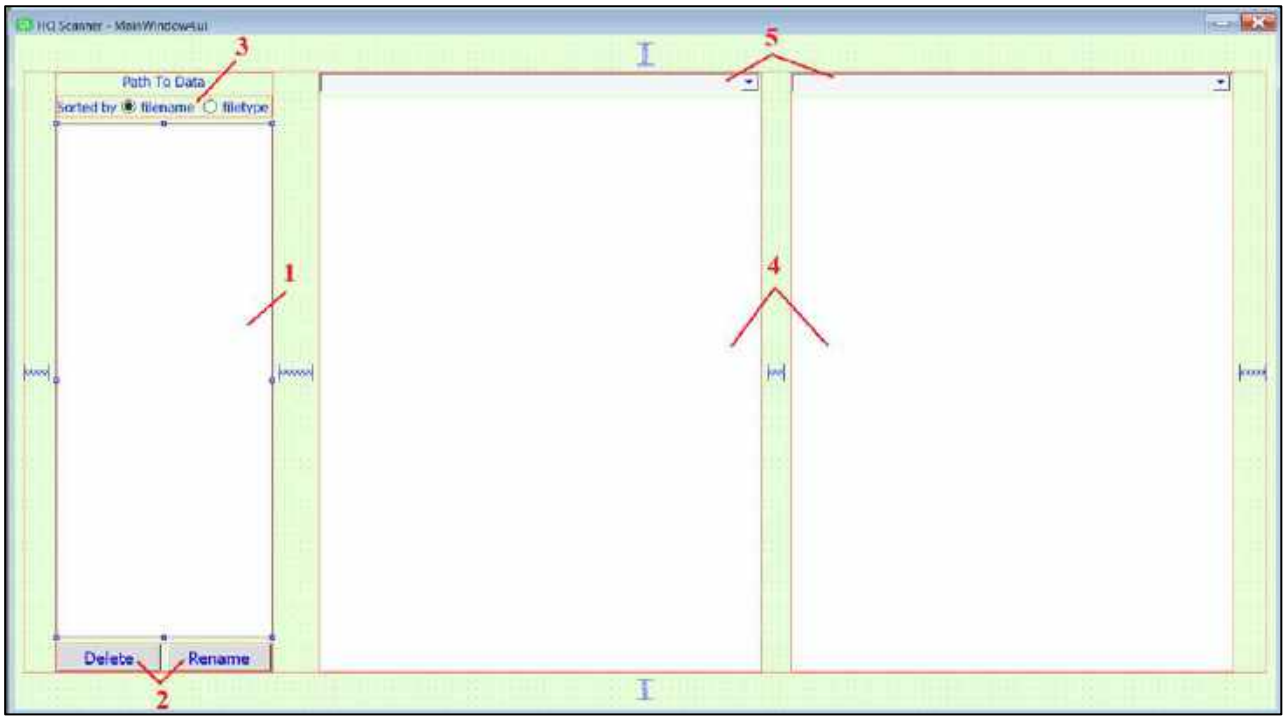


Рис. 5.3. Макет інтерфейсу головного вікна програми

Отриманий макет не відображає кінцевий варіант інтерфейсу головного вікна, а лише задає шаблон його побудови. Що стосуються наповнення макета необхідним функціоналом, то для цього отриманий файл інтерфейсу з розширенням «.ui» був переформатований у «.py» для подальшої роботи.

Після проведення програмних модифікацій макета головного вікна був сформований кінцевий варіант головного вікна, поданий на рис. 5.4.

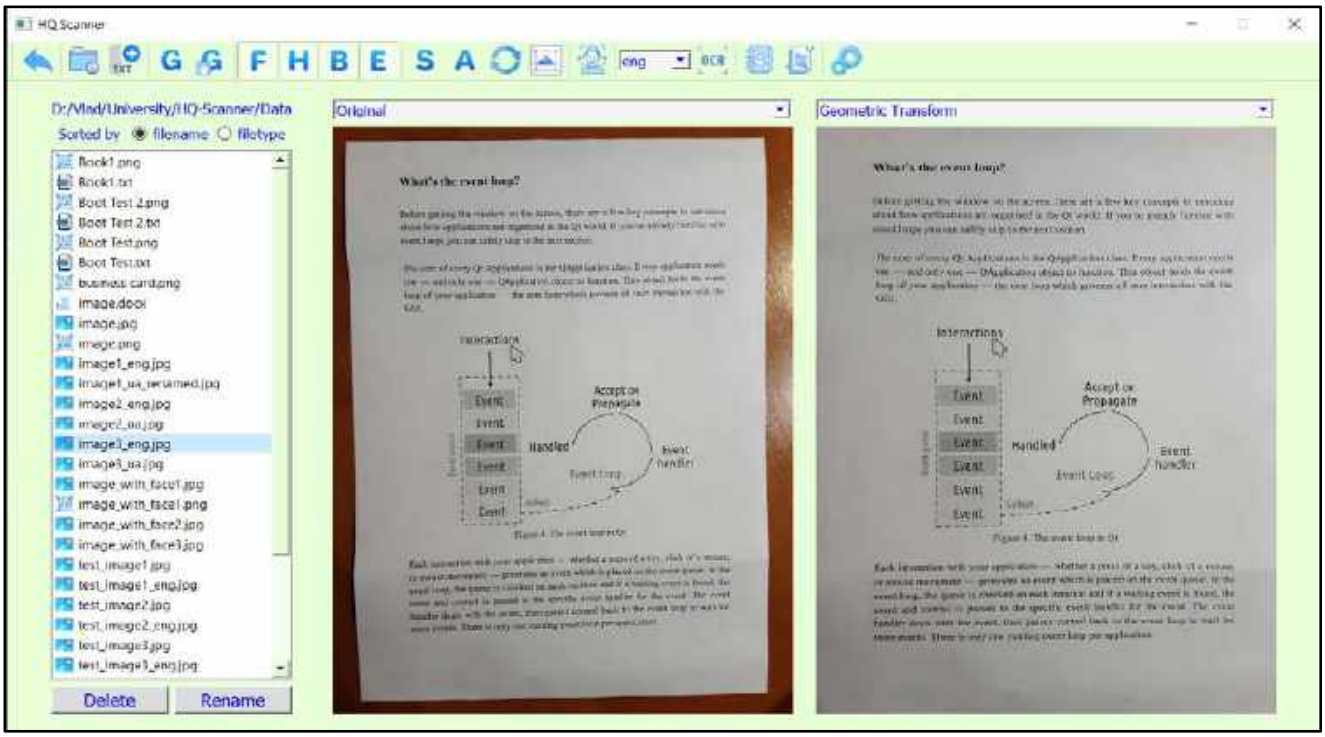
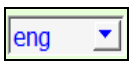


Рис. 5.4. Інтерфейс головного вікна програми

У верхній частині головного вікна було додано панель інструментів. Опис цілей кожної з іконок наведено в табл. 5.4.

Таблиця 5.4

Панель інструментів головного вікна

№ п/п	Іконка	Призначення
1		Повернення до вікна запуску програми
2		Зміна поточного каталогу даних
3		Створення нового текстового альбому
4		Виконання автоматичного алгоритму геометричних перетворень
5		Виконання інтерактивного алгоритму геометричних перетворень
6		Включення або виключення етапу фільтрації шумів
7		Включення або виключення етапу підкреслення меж
8		Включення або виключення етапу бінаризації
9		Включення або виключення етапу ерозії
10		Виконання операції сегментації
11		Виконання операції анонімізації
12		Очищення результатів сегментації та анонімізації
13		Виконання операції виділення та збереження рисунків
14		Виконання розпізнавання облич на поточному зображенні
15		Випадний список для вибору мови тексту під час розпізнавання
16		Виконання оптичного розпізнавання тексту
17		Створення візитної картки користувача у вигляді QR-коду
18		Декодування текстової інформації із зображення-обкладинки
19		Виклик вікна настройок програми

Як можна побачити із рис. 5.4, у результаті було реалізовано такий функціонал системи:

- відображення даних із каталогу в підвікні;
- відображення зображень на різних етапах підготовки з можливістю вибору етапу у випадному списку;
- оброблення зміни поточного зображення в підвікні файлів каталогу;
- можливість зміни режиму сортування файлів у каталозі за допомогою перемикача (radio button) «`filename`» і «`filetype`»;
- оброблення натискань кнопок видалення і перейменування.

Окрім головного вікна програми, у системі також використовуються допоміжні вікна:

1. Вікно запуску програми (рис. 5.5), яке використовується під час першого запуску та дає змогу почати (кнопка `Start`) або завершити (кнопка `Exit`) роботу програми. Також на це вікно повертає іконка № 1 (див. табл. 5.4) на панелі інструментів головного вікна.

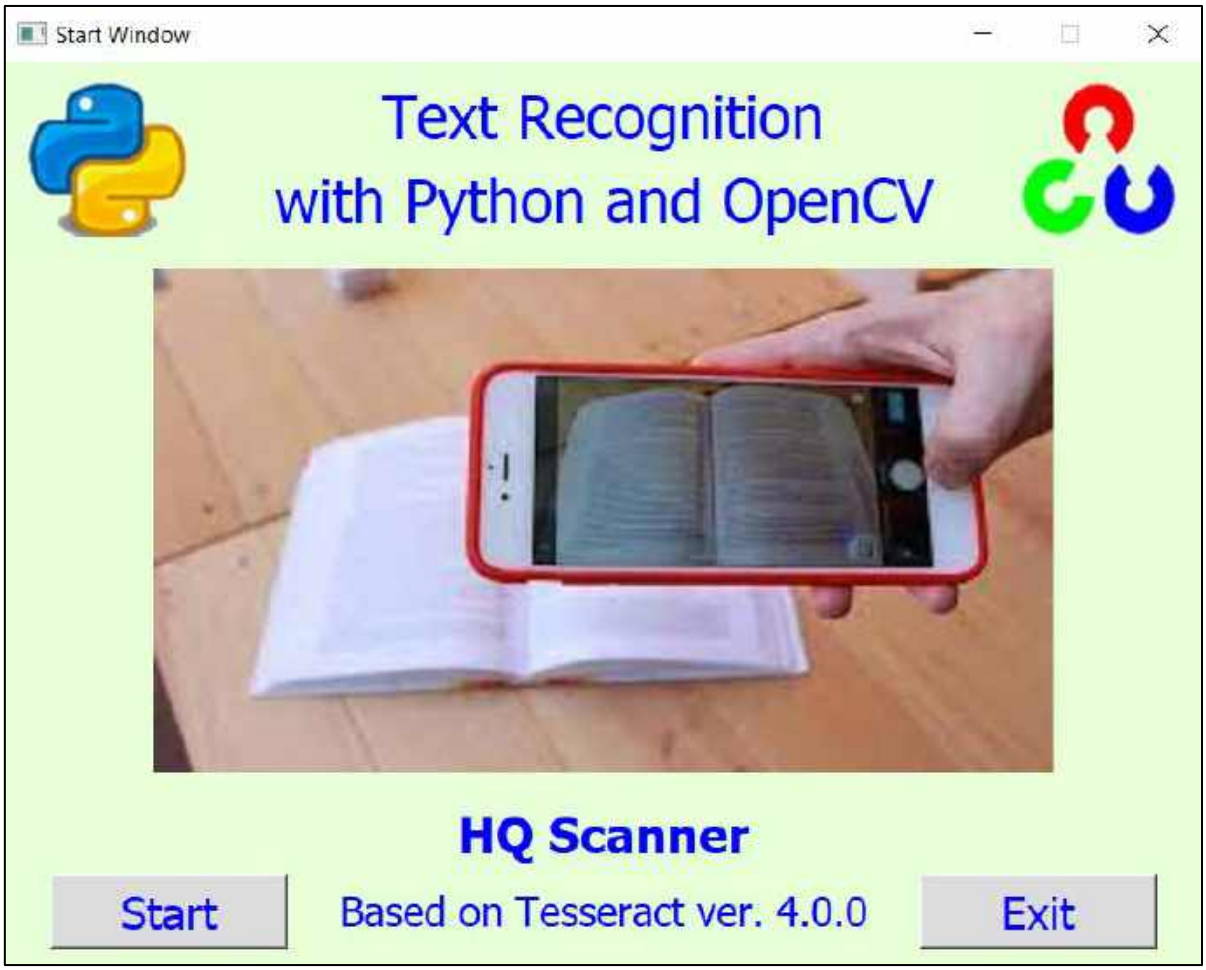


Рис. 5.5. Вікно запуску програми

Вікно запуску програми є її логотипом. У цьому вікні відображено суть програми (Text recognition with Python and OpenCV) та вказано версію її інформаційного ядра (Tesseract ver. 4.0.0). Розробники дали програмі назву HQ Scanner. Далі будемо використовувати це ім'я.

2. Вікно попереднього перегляду тексту (рис. 5.6), яке викликається після виконання оптичного розпізнавання тексту через іконку № 16 або декодування зображення-обкладинки № 18 (див. табл. 5.4). Має власну панель інструментів, опис якої наведено в табл. 5.5.

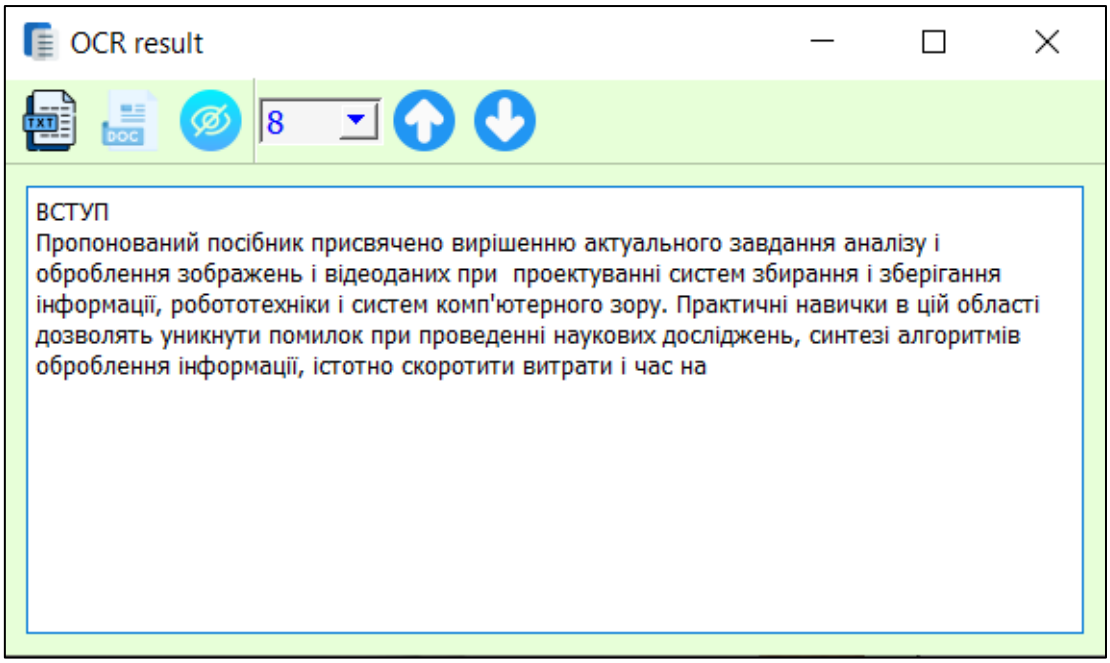


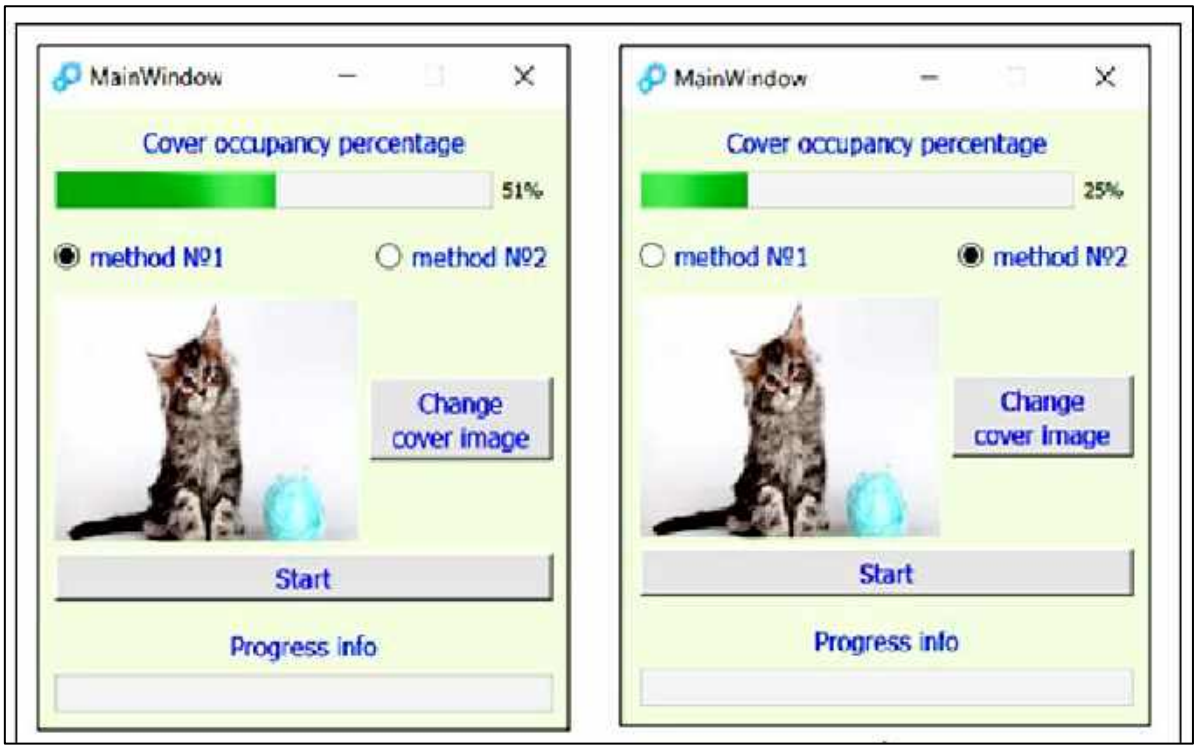
Рис. 5.6. Вікно попереднього перегляду тексту

Таблиця 5.5

Панель інструментів вікна попереднього перегляду тексту

№ п/п	Іконка	Призначення
1		Збереження тексту в альбомі у форматі txt
2		Збереження тексту у форматі doc
3		Кодування тексту в QR-код з подальшим виконанням стеганографії
4	8	Випадний список для вибору розміру шрифту у вікні
5		Збільшення розміру шрифту у вікні на один крок
6		Зменшення розміру шрифту у вікні на один крок

3. Вікно виконання стеганографії (рис. 5.7), яке викликається з вікна попереднього перегляду тексту після натискання іконки № 3 (див. табл. 5.5). У цьому вікні користувач може вибрати бажаний метод стеганографії, у разі потреби змінити зображення-обкладинку та побачити процентне значення наповненості зображення-обкладинки відповідно до текстової інформації, яку необхідно закодувати всередину.



а *б*

Рис. 5.7. Вікно виконання стеганографії

4. Вікно для створення візитної картки користувача (рис. 5.8), яке викликається після натискання іконки № 17 (див. табл. 5.4) у головному вікні програми.

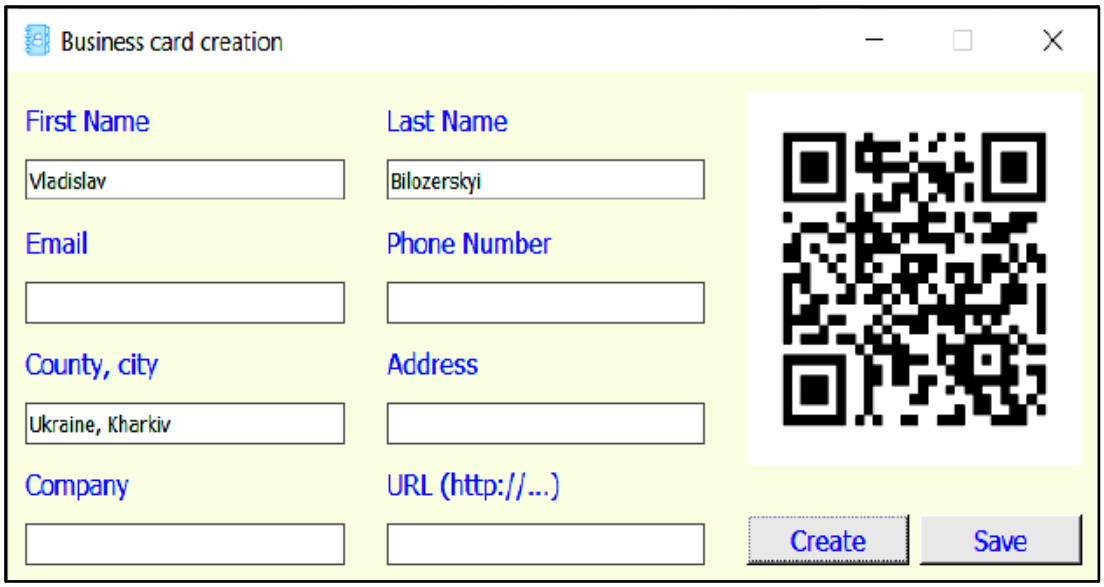


Рис. 5.8. Вікно створення візитної картки

5. Вікно відображення результатів виділення облич (рис. 5.9), яке викликається після натискання іконки № 14 (див. табл. 5.4) у головному вікні програми. Має панель інструментів з іконкою для збереження отриманих результатів і прапорцем для включення/виключення відображення виділених областей очей на зображенні.

Рис. 5.9. Вікно відображення результатів виділення облич

6. Вікно настройок (рис. 5.10), що викликається після натискання іконки № 19 (див. табл. 5.4) у головному вікні програми. Має панель інструментів з іконками для збереження поточних настройок та для повернення всіх значень за замовчуванням.

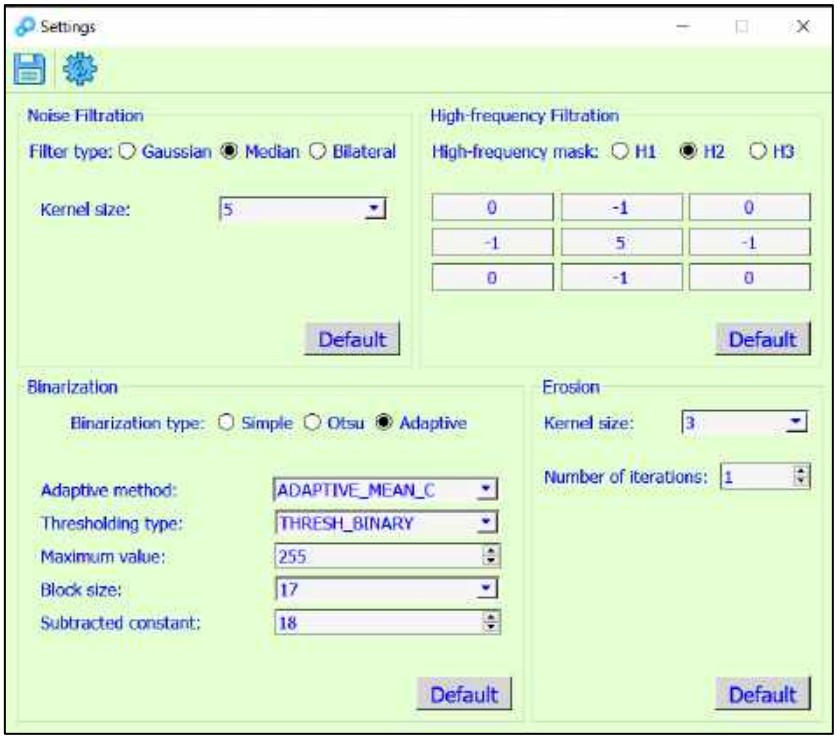


Рис. 5.10. Вікно настройок

Отже, сформовано програмний інтерфейс для системи оптичного розпізнавання тексту, який складається із:

- головного вікна, у якому користувач має доступ до використання всіх функціональних можливостей системи;
- шести допоміжних вікон, які реалізують додатковий функціонал відображення результатів певних операцій і дають змогу не перезавантажувати головне вікно для користувача.

5.3. Опис і підготовка до роботи програми HQ Scanner

Програма HQ Scanner – це результат інтеграції роботи рушія оптичного розпізнавання тексту Tesseract, розробленого для триетапного алгоритму підготовки зображення до розпізнавання (геометричні перетворення, попереднє оброблення тексту, сегментація та анонімізація), додаткових сервісних функцій і наочного й зрозумілого програмного інтерфейсу.

Використання цієї програми дає користувачам не тільки можливість оптичного розпізнавання тексту на зображеннях, взятих із різноманітних каталогів даних, але й глибше зрозуміти основні недоліки системи оптичного розпізнавання тексту під час оброблення зображень із нерівномірним освітленням, кутовими відхиленнями, шумами тощо. Це реалізовано завдяки таким особливостям програми:

- наявність одразу двох полів зображень у головному вікні програми із можливістю вибору відображення результатів конкретного етапу оброблення для кожного з них − така особливість дає змогу побачити як та чи інша операція впливає на зображення загалом;
- гнучкість настройок процесу попереднього оброблення тексту з можливістю вибору різних типів фільтрів, алгоритмів бінаризації та їх окремих параметрів, також при цьому можна включати і виключати кожну із чотирьох операцій оброблення, змінюючи при цьому загальний алгоритм;
- наявність як автоматичного алгоритму геометричних перетворень, так і інтерактивного режиму з виділенням кутових точок текстового документа для зображень, де автоматичний алгоритм не дає задовільного результату;
- наявність операцій сегментації та анонімізації зображення для концентрації уваги під час розпізнавання лише на необхідних користувачу фрагментах тексту та захисту анонімних фрагментів зображення, які мають бути нерозпізнаними;
- можливість швидкого збереження результатів розпізнавання у форматах `pdf` і `doc`, а також є можливим кодування розпізнаного тексту в QR-код із подальшим збереженням у форматі `png` для забезпечення захисту інформації, при цьому можна швидко відкривати сформовані файли безпосередньо із головного вікна програми.

Підготовка до роботи. Для можливості використання програми HQ Scanner на пристрої необхідно виконати такі дії:

1. Установити Python на пристрій, якщо така програма не встановлена. Найкращий спосіб встановити Python на Windows – це завантажити офіційний інсталятор із сайту `python.org`. Для цього достатньо відкрити браузер, перейти за посиланням https://python.org/ і натиснути на вкладку *Downloads* (рис. 5.11). Сайт виявить, що ви відвідали його з Windows, і запропонує завантажити останню версію Python 3 або Python 2. Програма HQ Scanner розроблена на третій версії Python, тому завантажити необхідно саме її. Під вкладками *Downloads* → *Download for Windows* треба натиснути кнопку *Python 3.XX* (рис. 5.12), щоб завантажити інсталяційний файл. Після завантаження слід натиснути на інсталятор і виконати необхідні інструкції, після чого перезавантажити пристрій.

2. Установити спеціальні бібліотеки Python, які використовуються в програмі. Найдоступнішим методом встановлення бібліотек для Python є використання `pip` – менеджера пакетів для Python, робота з яким здійснюється за допомогою командного рядка. З Python версії 3.4 `pip` поставляється разом з інтерпретатором Python, тому його непотрібно встановлювати окремо. Найперше необхідно відкрити командний рядок на пристрої. Для цього в пошуку Windows треба записати текст «`cmd`» і натиснути «`enter`» (рис. 5.13). У відкритому командному рядку необхідно ввести спеціальну команду для встановлення кожної з бібліотек по черзі (рис. 5.14):

1. «`pip install -upgrade pip`» – оновлення pip до останньої версії;
2. «`pip install numpy`» – бібліотека NumPy;
3. «`pip install opencv-python`» – бібліотека OpenCV;
4. «`pip install imutils`» – бібліотека Imutils;
5. «`pip install Pillow`» – бібліотека Pillow;
6. «`pip install pytesseract`» – бібліотека pytesseract;
7. «`pip install exifread`» – бібліотека exifread;
8. «`pip install fpdf`» – бібліотека fpdf;
9. «`pip install python-docx`» – бібліотека docx;
10. «`pip install PyQt5`» – бібліотека PyQt5

Рис. 5.11. Вкладка Downloads на офіційному сайті Python

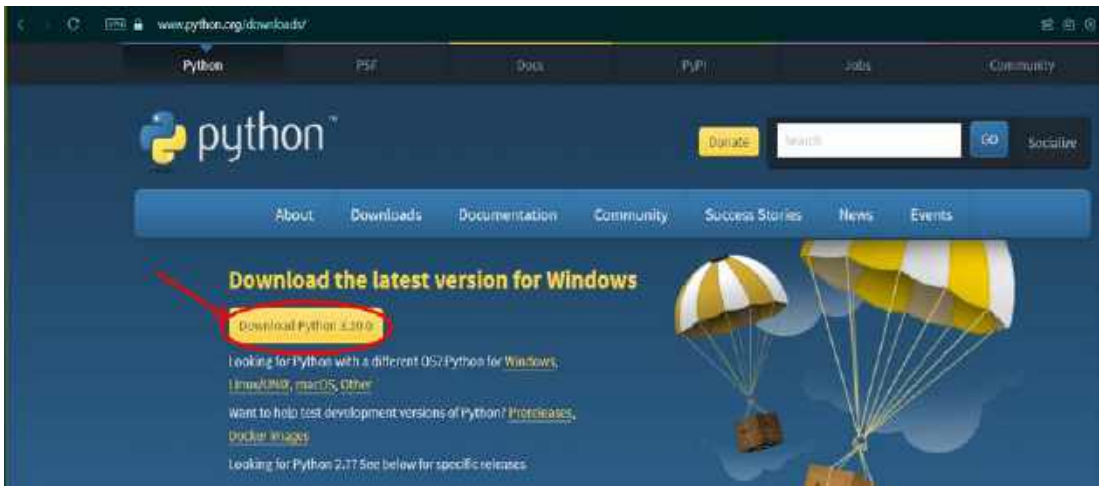


Рис. 5.12. Завантаження останньої версії Python 3.x

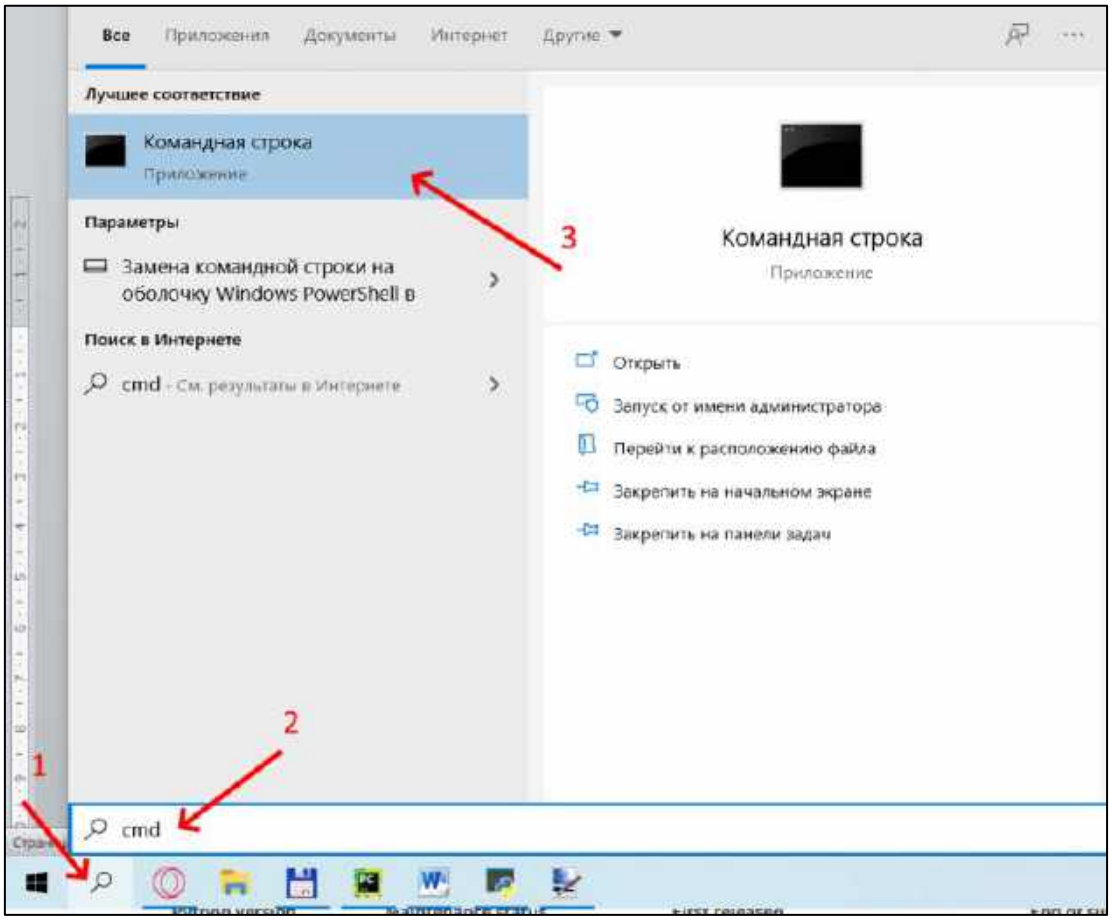


Рис. 5.13. Відкриття командного рядка

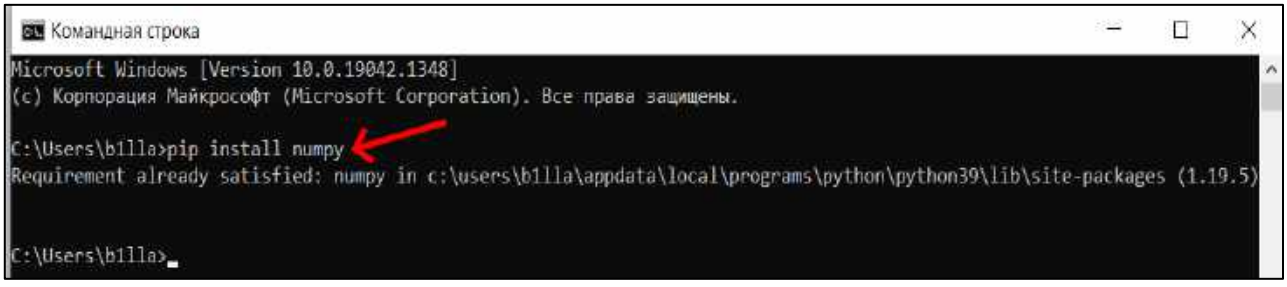


Рис. 5.14. Приклад встановлення бібліотеки за допомогою командного рядка

3. Установити рушій оптичного розпізнавання тексту Tesseract на пристрій. Для цього перейти на офіційний сайт із документацією за посиланням https://tesseract-ocr.github.io/tessdoc/Downloads.html та вибрати `Releases` у вкладці `Source Code` (рис. 5.15). Після цього в репозиторії `github` можна завантажити останню версію рушія у вигляді zip-архіву (рис. 5.16). Останній етап – розпакувати завантажений архів до каталогу з програмою.

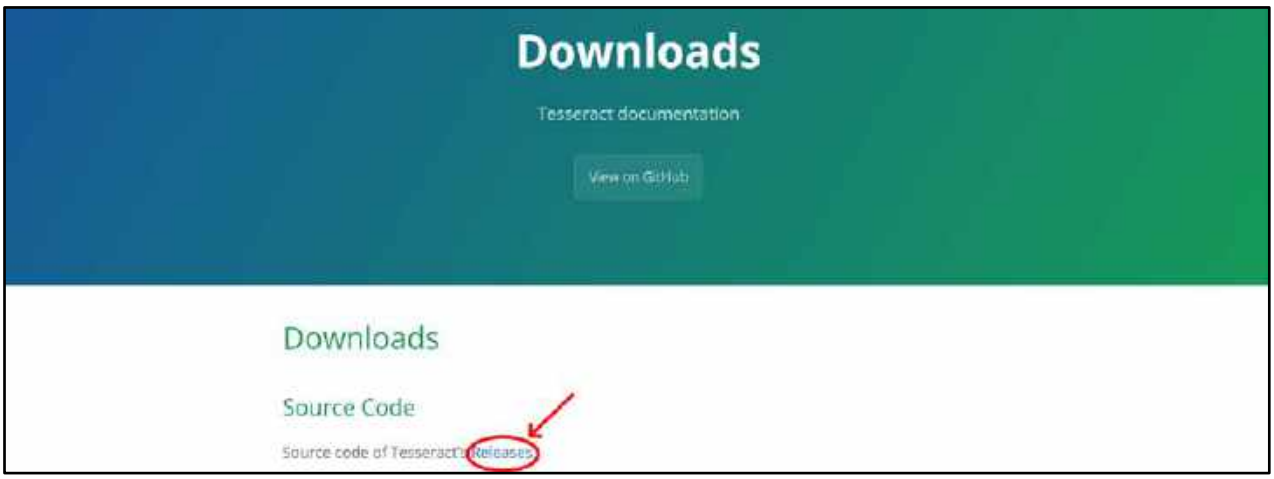


Рис. 5.15. Перехід до Releases на сайті документації

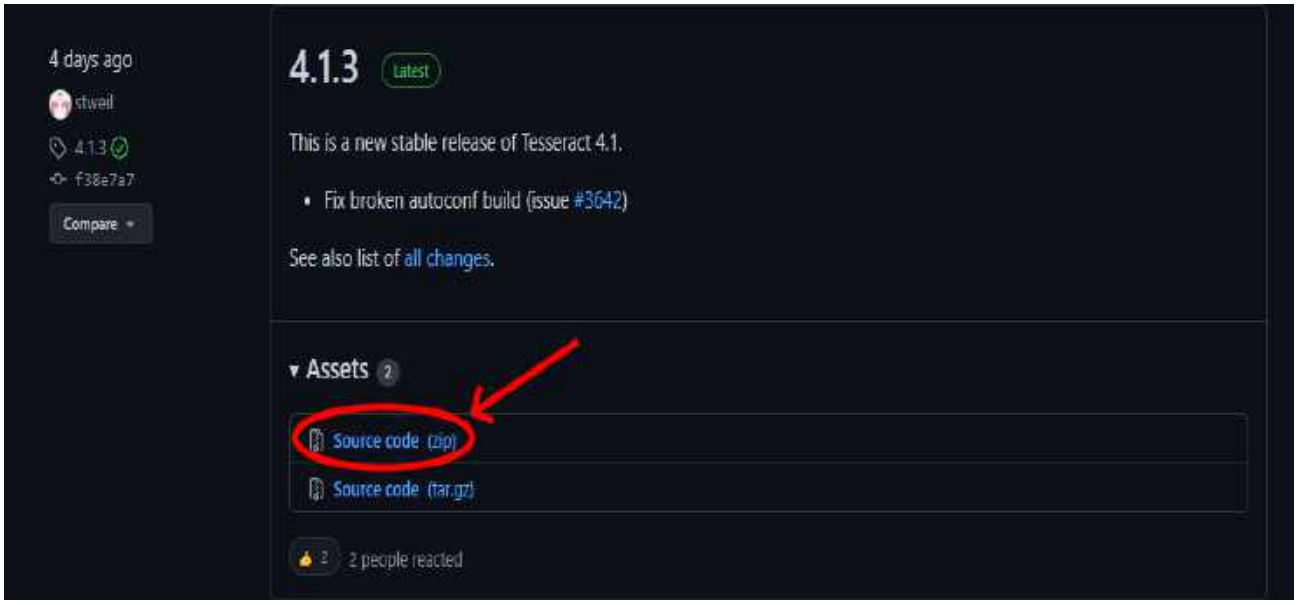


Рис. 5.16. Завантаження останньої версії рушія

Виконання цих етапів завершує попередню підготовку до роботи та дає змогу виконати перший запуск програми HQ Scanner.

5.4. Робота з програмою HQ Scanner

Під час запуску програми HQ Scanner користувач бачить стартове вікно (рис. 5.17), на якому наявні дві основні кнопки для взаємодії – «`Start`» для початку роботи та «`Exit`» для завершення роботи і закриття стартового вікна.

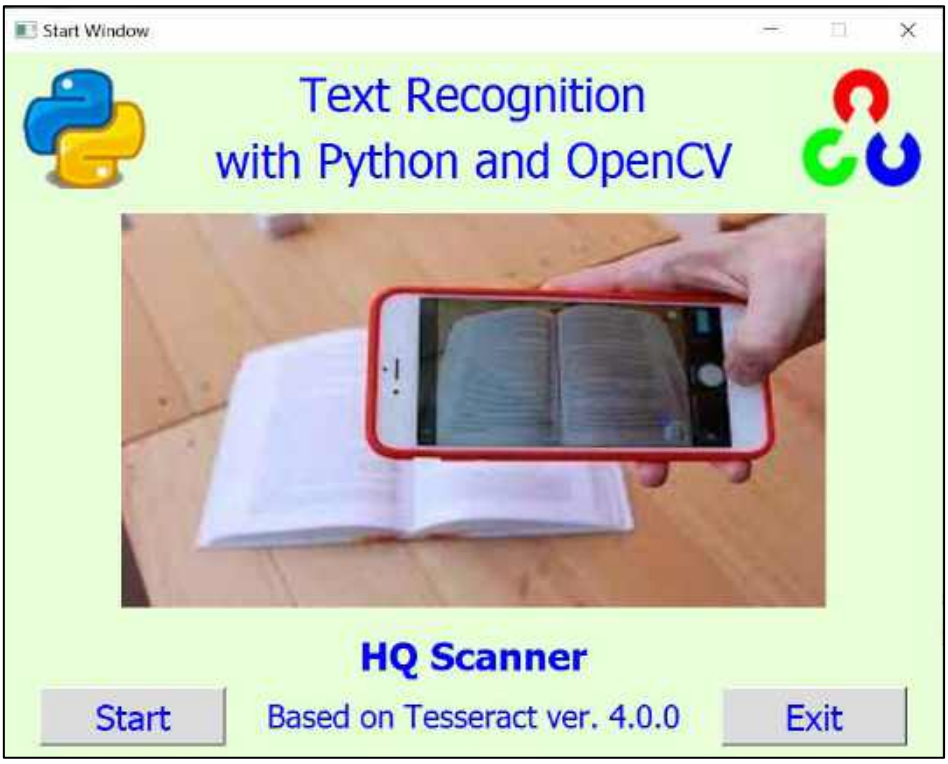


Рис. 5.17. Стартове вікно програми

Робота кнопки «`Exit`» не потребує додаткових пояснень, тому слід розглянути детальніше «`Start`». Так, після натискання цієї кнопки відкривається діалогове вікно пошуку каталогу даних для подальшої роботи (рис. 5.18). Користувач може вибрати будь-який каталог на пристрої або активних носіях інформації та вибрати в каталозі конкретне зображення у форматі `jpg`. Це приведе до закриття стартового вікна та відкриття головного вікна програми з вибраним зображенням як поточним (рис. 5.19).

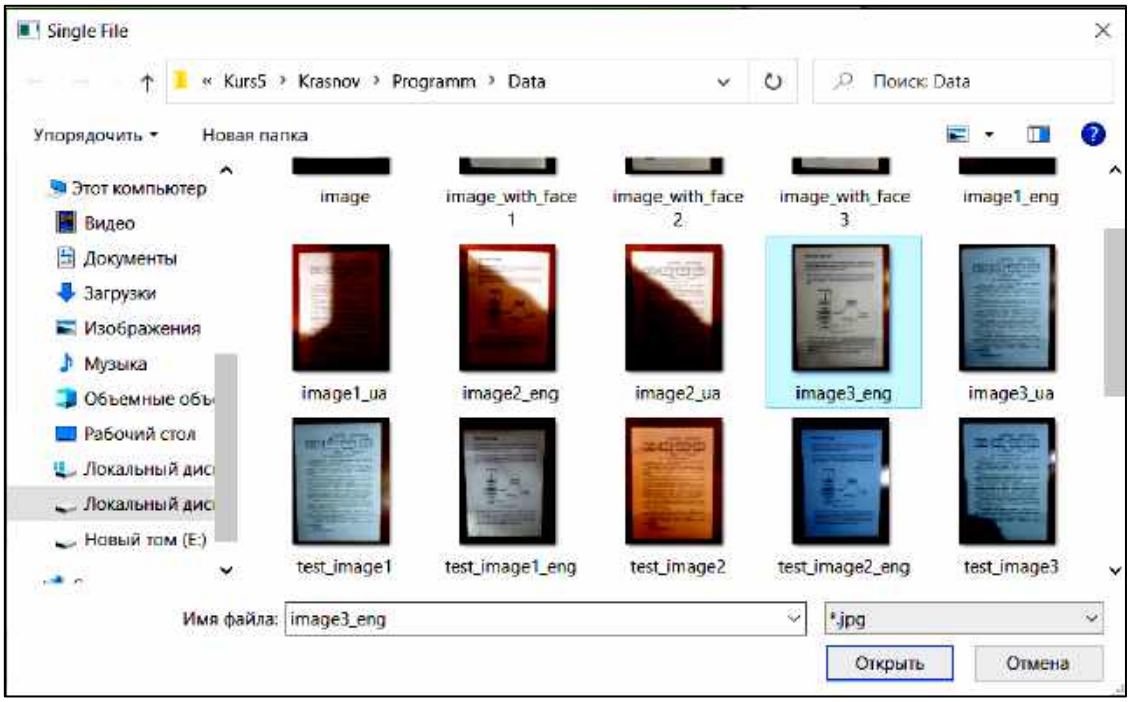


Рис. 5.18. Діалог пошуку каталогу даних

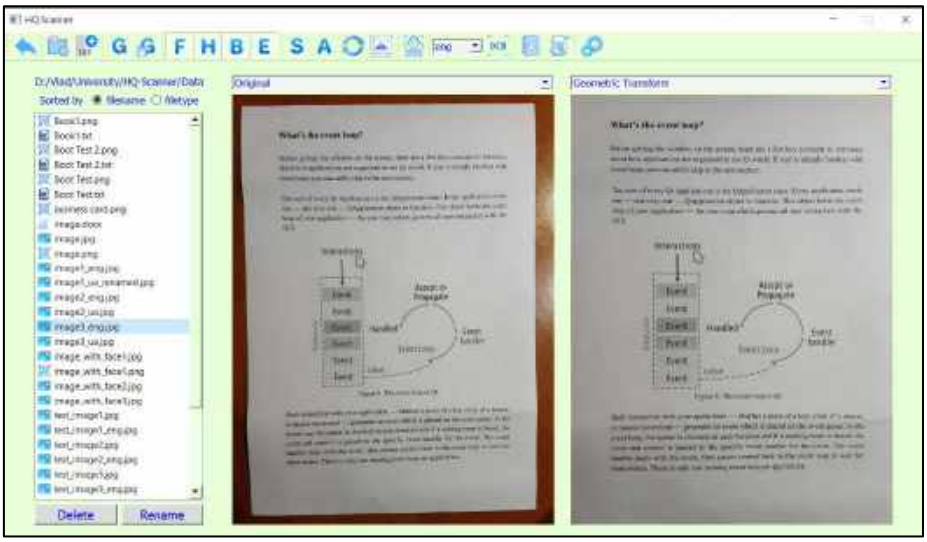


Рис. 5.19. Головне вікно програми

Функціонал головного вікна програми. Головне вікно програми має широкий арсенал можливостей, які надані користувачу. З метою уникнення можливих непорозумінь користувача під час роботи, треба розглянути кожну функцію на конкретному прикладі.

Використання випадних списків, розташованих над полями зображень, для зміни поточного етапу оброблення зображення для відображення результату. Обидва списки мають однакові елементи, які відповідають кожному з етапів підготовки зображення перед розпізнаванням, і дають змогу продивитися результати кожного з етапів підготовки зображення. Для цього досить просто натиснути на один зі списків і вибрати необхідний етап оброблення, після чого зображення під списком зміниться відповідно до вибраного етапу. Приклад застосування наведено на рис. 5.20.

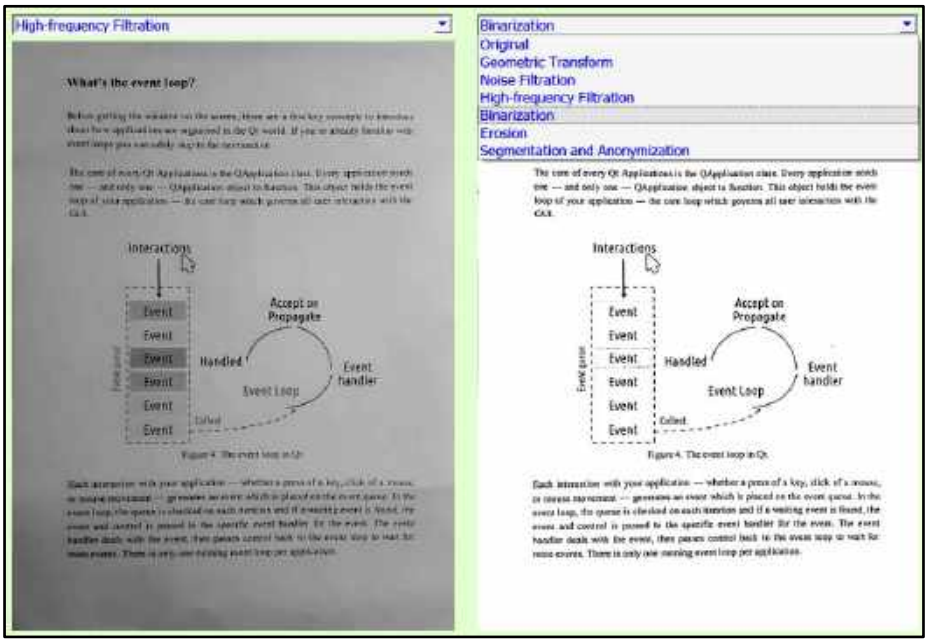


Рис. 5.20. Зміна відображення першого зображення з `Original` на `High-frequency Filtration` та другого на `Binarization`

Зміна поточного зображення. Для цього необхідно в підвікні перегляду файлів каталогу вибрати будь-яке зображення формату jpg (окрім поточного), що приведе до оновлення всієї інформації, що відображена у вікні, відповідно до нового зображення. Також важливо зазначити, що зміна поточного зображення веде до автоматичного виконання етапів геометричних перетворень і попереднього оброблення тексту для нового. Приклад застосування подано на рис. 5.21.

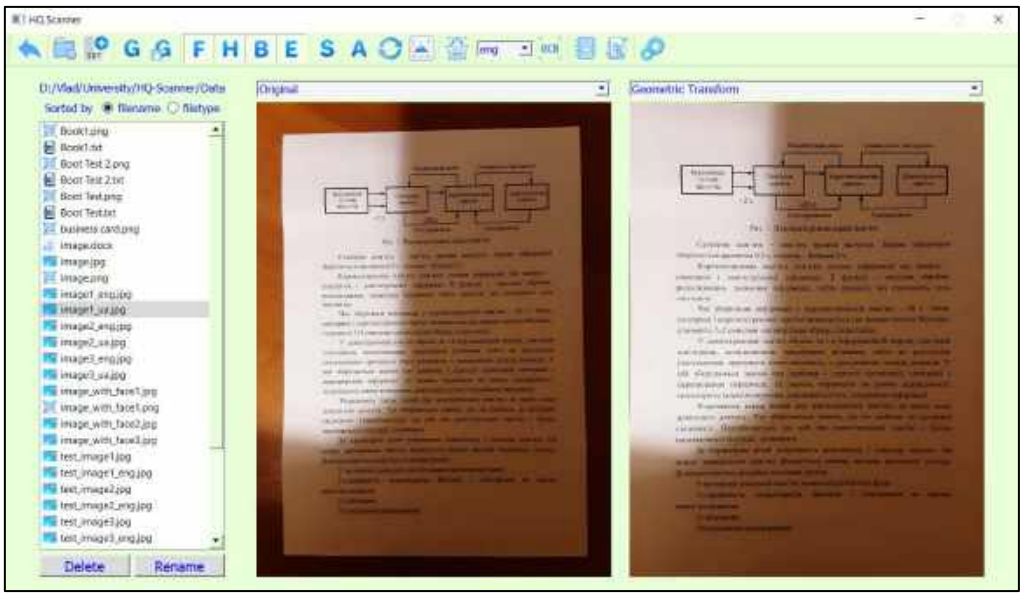


Рис. 5.21. Зміна поточного зображення на `image1_ua.jpg`

Видалення файлів із каталогу даних. Ця функція реалізується натисканням кнопки «Delete», що розташована під вікном відображення файлів каталогу даних. Після натискання на кнопку поточний файл видаляється з каталогу та не може бути вибраний для роботи. Приклад застосування наведено на рис. 5.22.

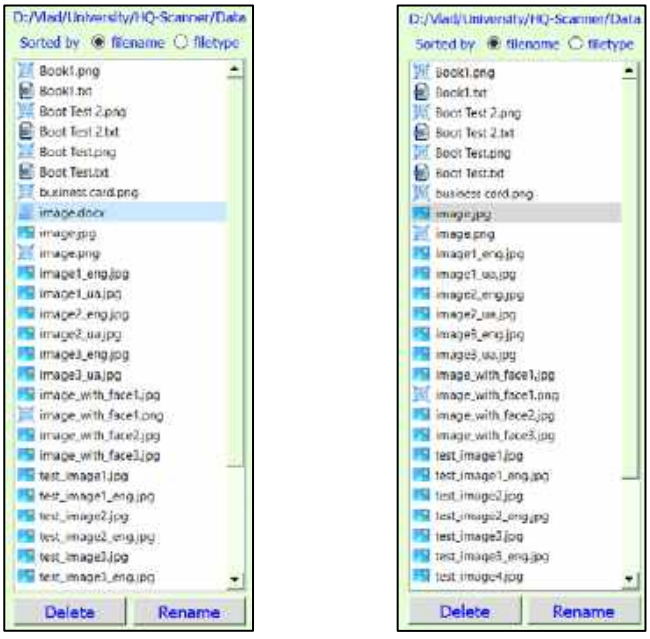


Рис. 5.22. Видалення файлу `image.docx`

Перейменування файлів із каталогу даних. Ця функція реалізується натисканням кнопки «Rename», що розташована під вікном відображення файлів каталогу даних. Після натискання на кнопку відкривається діалогове вікно, у якому необхідно заповнити поле нового імені файлу та натиснути «`Apply`» для перейменування або «`Cancel`» для припинення. Приклад застосування показано на рис. 5.23.

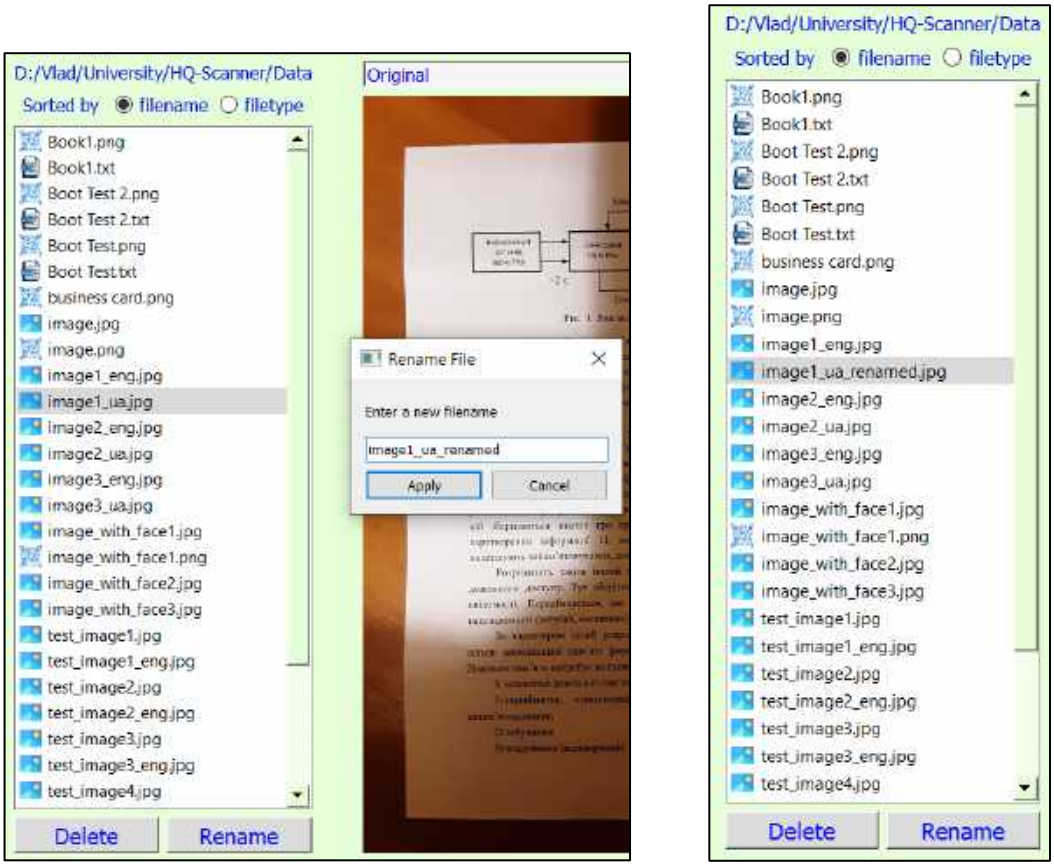


Рис. 5.23. Перейменування файлу `image1_ua.jpg` на `image1_ua_renamed.jpg`

Перегляд файлів форматів `*txt`, `*doc` та `*png`. У разі зміни поточного елемента в підвікні перегляду файлів програма реагує лише на файли формату `*jpg` (як зображення для оброблення) та ігнорує файли інших форматів. Але для файлів форматів `*doc` та `*png` передбачена додаткова функція їх відкриття у середовищах за замовчуванням на пристрої (файли doc відкриються в Microsoft Word). Для цього необхідно виконати подвійне натискання на ім'я відповідного файлу в підвікні перегляду файлів. Для файлів формату `*txt` у разі подвійного натискання програма відкриває вікно попереднього перегляду тексту.

Зміна способу сортування файлів у каталозі даних. Ця функція реалізується за допомогою перемикачів (radio boitton) «`filename`» і «`filetype`». У разі активного стану кнопки «`filename`» (режим за замовчуванням) файли сортуються за алфавітом відповідно до їх імені, а в разі активного стану кнопки «`filetype`» – відповідно до типу даних. Приклад застосування подано на рис. 5.24.

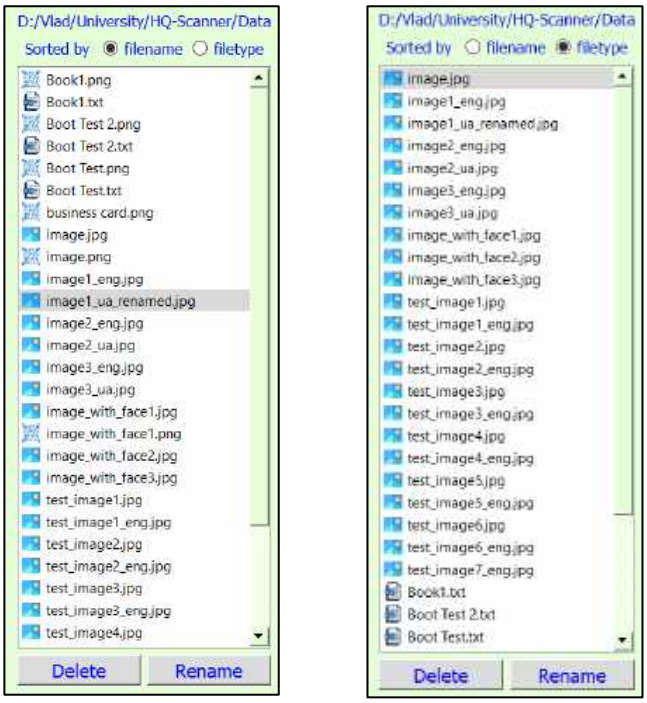


Рис. 5.24. Зміна способу сортування файлів у каталозі даних

Повернення до стартового вікна програми (див. рис. 5.17). Для виконання необхідно натиснути на іконку № 1 (див. табл. 5.4), розташовану на панелі інструментів головного вікна програми.

Зміна поточного каталогу. Для виконання необхідно натиснути на іконку № 2 (див. табл. 5.4), розташовану на панелі інструментів головного вікна програми. Після натискання відкриється діалогове вікно пошуку (див. рис. 5.18), у якому можна вибрати новий каталог із даними для роботи.

Створення нового альбому для збереження текстових даних. Для виконання необхідно натиснути на іконку № 3 (див. табл. 5.4) та в діалоговому вікні, що відкрилося (рис. 5.25), записати ім’я нового альбому.

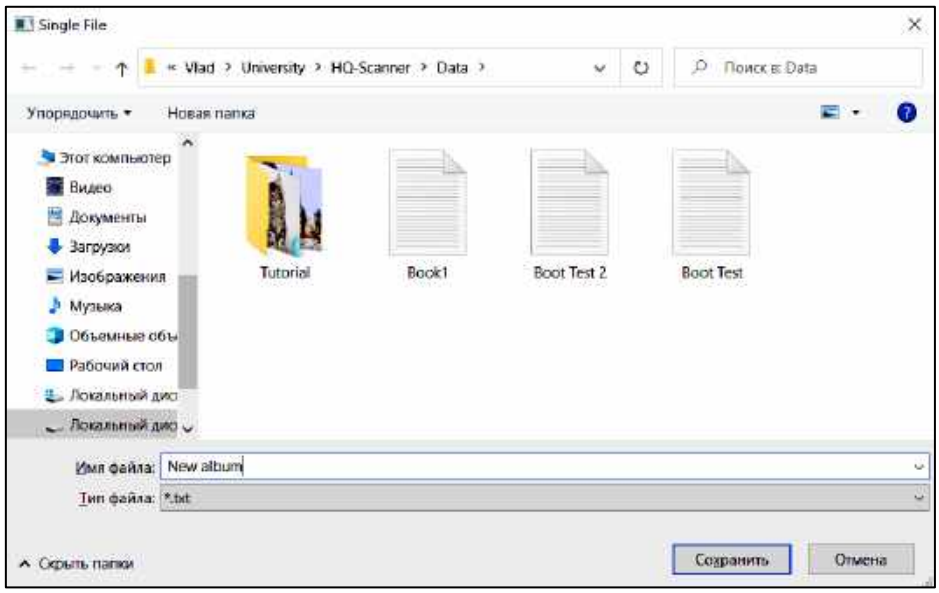


Рис. 5.25. Діалогове вікно для створення текстового альбому

Виконання інтерактивного алгоритму геометричних перетворень поточного зображення. Реалізується натисканням на іконку № 5 (див. табл. 5.4), розташовану на панелі інструментів головного вікна програми. Після натискання відкриється вікно, у якому можна виділити чотири кутові точки текстового документа (рис. 5.26). У цьому вікні можна натискати клавіші «b» – для скасування останньої виділеної точки, «r» – для видалення всіх точок та «enter» – для збереження виділених точок. Якщо після натискання клавіші «enter» було виділено чотири точки на зображенні, то виконується перспективне перетворення за точками й отриманий результат буде відображено в другому полі перегляду зображень як результат етапу `Geometric Transform`. Оскільки виконання інтерактивного алгоритму змінює зображення, що відповідає за етап `Geometric Transform`, то після його виконання оновлюються зображення наступних етапів оброблення. Для скасування результатів інтерактивного режиму можна натиснути на іконку № 4 (див. табл. 5.4), у результаті чого виконається автоматичний алгоритм геометричних перетворень. За аналогією з інтерактивним режимом зміна зображення, що відповідає за етап `Geometric Transform`, приводить до оновлення зображення для всіх наступних етапів оброблення.

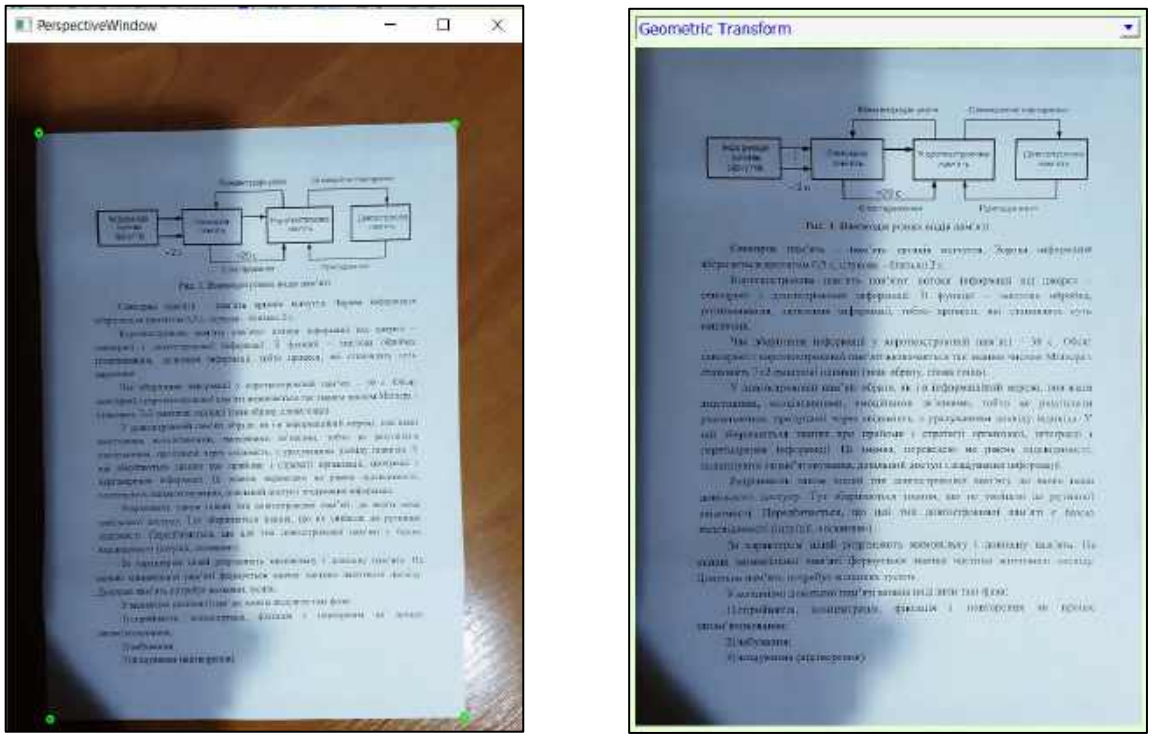
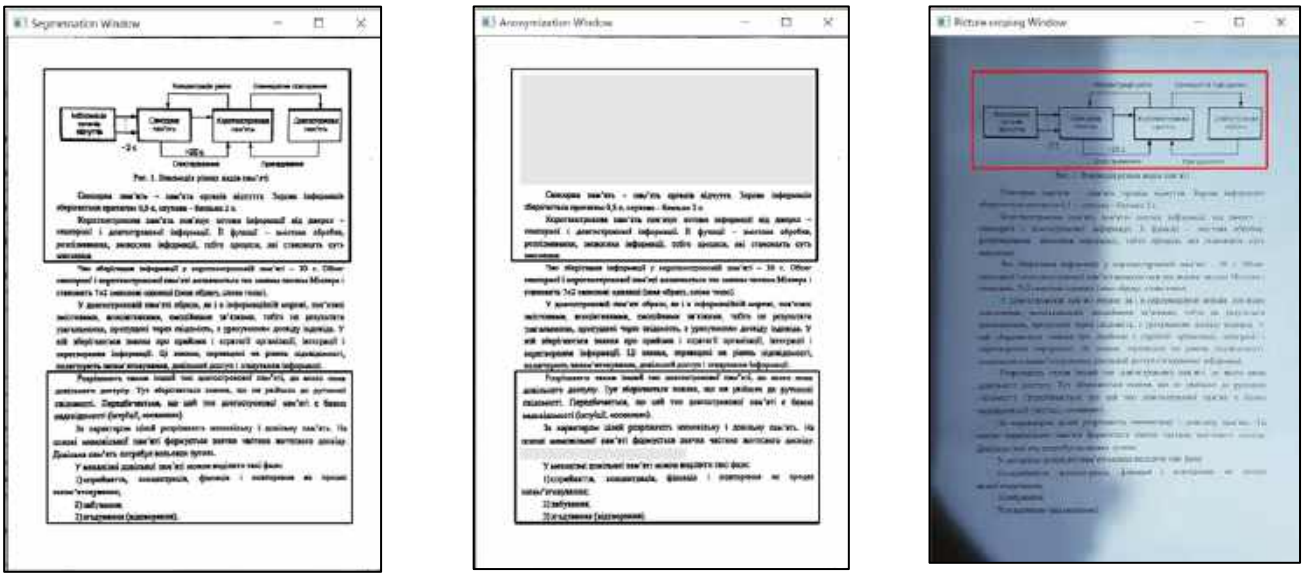


Рис. 5.26. Виконання інтерактивного алгоритму геометричних перетворень

Включення та виключення окремих етапів попереднього оброблення тексту. Іконки № 6, 7, 8, 9 (див. табл. 5.4), що розташовані на панелі інструментів головного вікна, являють собою кнопки з двома можливими станами (`On/Off`). При цьому кожна з іконок відповідає за певний етап попереднього оброблення тексту: фільтрацію шумів, підкреслення меж, біна-

ризацію та ерозію відповідно. Якщо іконка в стані «On», то відповідна операція оброблення виконується, а в стані «Off» − ігнорується. При цьому перемикання будь-якої іконки з одного стану в інший приводить до зміни зображення, що відповідає за результат цього етапу оброблення, унаслідок чого відбувається оновлення зображення для всіх наступних етапів оброблення. Завдяки використанню цих іконок можна виключати етапи попереднього оброблення, які користувач вважає зайвими для поточного зображення.

Виконання інтерактивних операцій сегментації, анонімізації та виділення рисунків на зображенні. Для кожної з наведених операцій використовуються іконки № 10, 11, 13 (див. табл. 5.4). Після натискання будь-якої з них відкривається вікно, у якому можна в реальному часі виділяти прямокутні фрагменти на зображенні. У режимі сегментації ці фрагменти використовуються як області інтересу (ROI – Region Of Interes) в разі подальшого розпізнавання тексту (рис. 5.27, *а*), у режимі анонімізації – зафарбовуються сірим кольором (рис. 5.27, *б*), у режимі виділення рисунків – перетворюються в окремі зображення для подальшого збереження (рис. 5.27, *в*). У кожному з вікон можна натискати клавіші «b» – для скасування останнього виділеного фрагмента, «r» – для видалення всіх фрагментів та «enter» – для збереження виділених фрагментів. Результати сегментації та анонімізації використовуються програмою як фінальний варіант зображення перед розпізнаванням тексту (рис. 5.28), тоді як результати виділення рисунків зберігаються в окремому каталозі «Pictures» у корені програми як зображення формату *png. Також можна очистити результати виділення фрагментів анонімізації та сегментації натисканням іконки № 12 (див. табл. 5.4).



а *б* *в*

Рис. 5.27. Виконання інтерактивних операцій: *а* – сегментації; *б* – анонімізації; *в* − виділення рисунків

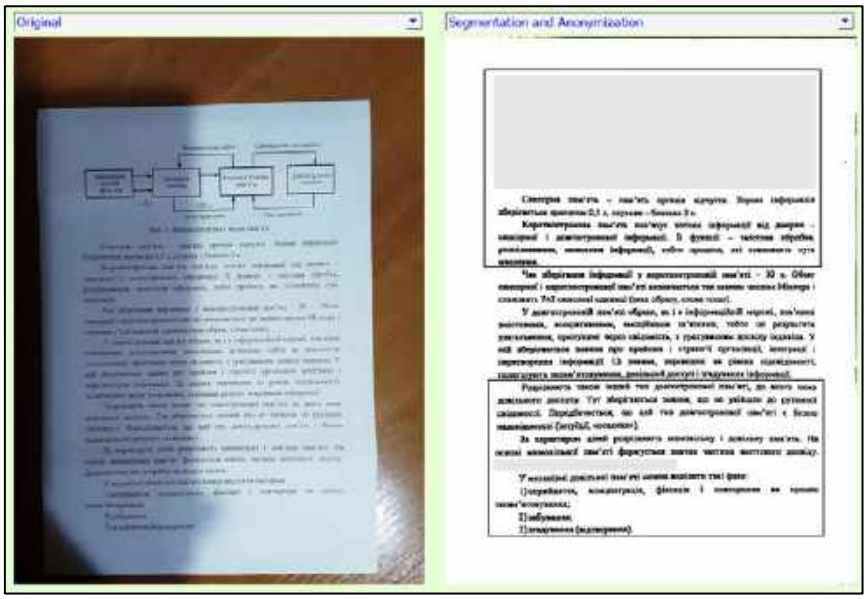


Рис. 5.28. Об’єднання результатів сегментації та анонімізації

Розпізнавання облич на зображенні. Реалізується натисканням на іконку № 14 (див. табл. 5.4), розташовану на панелі інструментів головного вікна програми. Після натискання відкриється вікно, у якому можна побачити результати виділення (рис. 5.29, *а*). Вікно з результатами має додаткову панель інструментів з іконкою для збереження виділеного фрагмента обличчя (за його наявності) у окремий каталог «`Face Detection Results`» як зображення у форматі `*png` та прапорця «`Eyes`» для перемикання режиму відображення результатів. Якщо прапорець активний, то на зображенні також відображаються виділені алгоритмом очі (рис. 5.29, *б*).

а *б*

Рис. 5.29. Результати виділення облич на зображенні

Виконання оптичного розпізнавання тексту. Реалізується натисканням на іконку № 16 (див. табл. 5.4), розташовану на панелі інструментів головного вікна програми. Після натискання починається процес розпізнавання тексту на фінальному варіанті поточного зображення після всіх етапів підготовки, у результаті якого відкривається вікно з результатами розпізнавання (рис. 5.30), можливістю редагування отриманого тексту та його збереження у форматах `*txt` і `*doc`. Для збереження у форматі `*txt` необхідно натиснути на іконку № 1 (див. табл. 5.5) додаткової панелі інструментів вікна, у форматі `doc` – на іконку № 2 (див. табл. 5.5). У разі натискання на іконку № 3 (див. табл. 5.5) відкривається вікно виконання стеганографії (див. рис. 5.7), у якому можна закодувати текстову інформацію в зображення `*png`. При цьому отримані файли будь-якого з трьох форматів зберігаються безпосередньо в каталог даних і можуть бути викликані подвійним натисканням у підвікні перегляду файлів. Також на додатковій панелі інструментів вікна результатів розпізнавання розташовані іконки № 5, 6, 7 (див. табл. 5.5), які використовуються для зміни розміру шрифту. Що стосується мови розпізнавання, то її можна задати у випадному списку № 15 (див. табл. 5.4) на панелі інструментів головного вікна. У цій реалізації програми можна вибрати українську, російську та англійську мови.

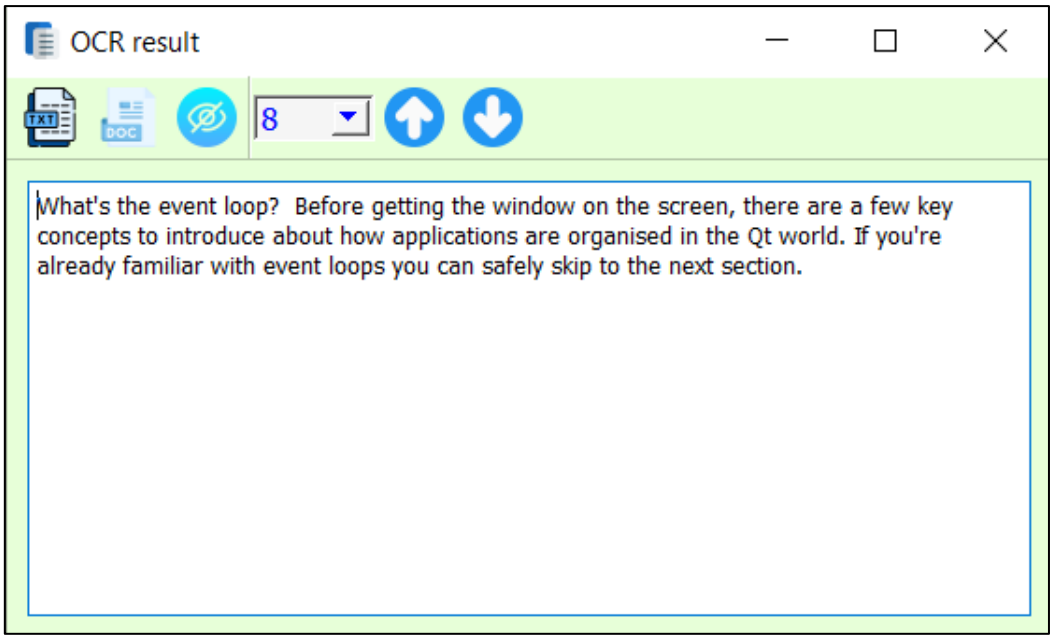


Рис. 5.30. Вікно з результатами розпізнавання тексту

Створення візитної картки. Реалізується натисканням на іконку № 17 (див. табл. 5.4), що розташована на панелі інструментів головного вікна програми. Після натискання відкривається відповідне вікно, у якому необхідно заповнити пусті поля з інформацією про користувача. Заповнення всіх полів не є обов'язковим. Після заповнення полів можна натиснути на кнопку «Create», яка сформує відповідний QR-код, у якому буде зашифровано особисту інформацію користувача (рис. 5.31). Також сформова-

ний QR-код можна зберегти в поточному каталозі даних за допомогою кнопки «Save». Зображення з QR-кодом зберігається у форматі *png та з конкретним ім'ям файлу – `business card`.

Рис. 5.31. Створення візитної картки користувача

Декодування текстової інформації із зображення-обкладинки. Реалізується натисканням на іконку № 18 (див. табл. 5.4), що розташована на панелі інструментів головного вікна програми. Після натискання відкривається діалогове вікно пошуку зображень у форматі *png (рис. 5.32) для виконання декодування текстової інформації із зображення. Після цього відкривається розглянуте раніше вікно результатів розпізнавання тексту (див. рис. 5.30), у якому можна побачити декодований текст.

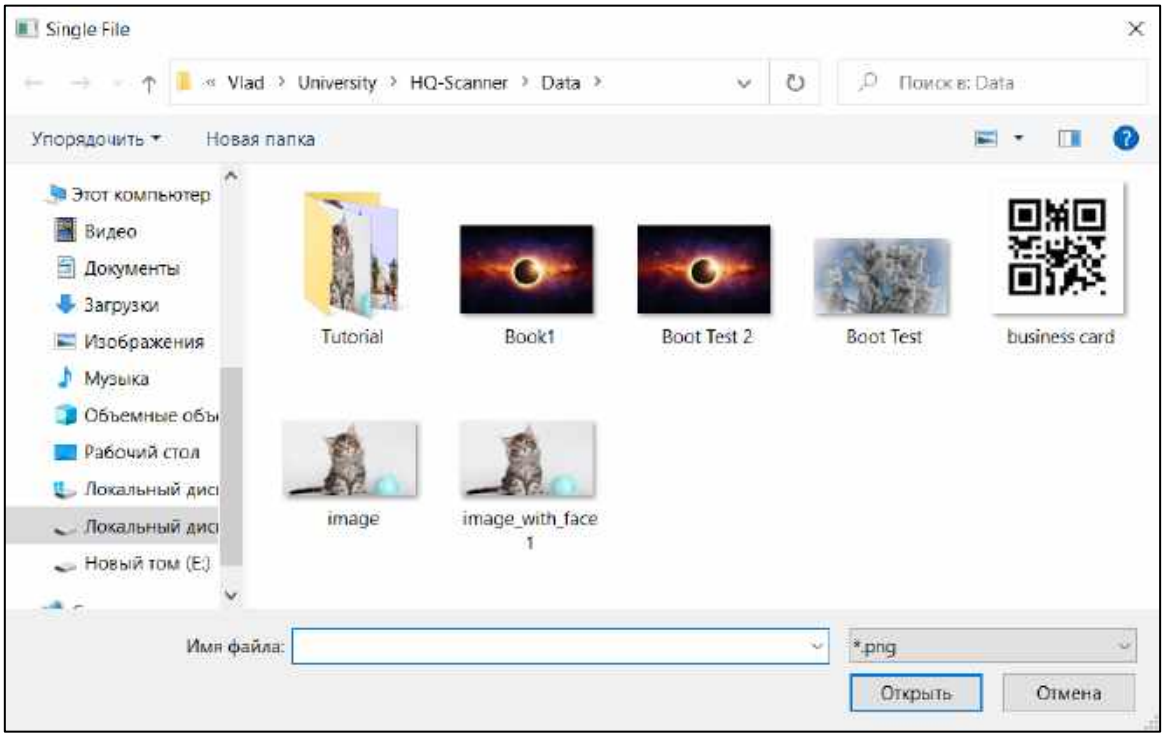


Рис. 5.32. Діалогове вікно для пошуку зображення-обкладинки у форматі png

Настроювання параметрів попереднього оброблення тексту. Реалізується натисканням на іконку № 19 (див. табл. 5.4), що розташована на панелі інструментів головного вікна програми. Після натискання відкривається вікно настройок (див. рис. 5.10), у якому можна коригувати параметри кожного із чотирьох етапів попереднього оброблення тексту: фільтрації шумів, підкреслення меж, бінаризації та ерозії. Оновленні параметри можна зберегти або встановити значення за замовчуванням для всіх етапів.

Таким чином, у цьому розділі наведено розгорнутий опис структури програми системи оптичного розпізнавання текстів HQ Scanner, докладний опис використовуваних класів, атрибутів і методів. Також докладно описано інтерфейс та основний функціонал програмного продукту.

За результатами досліджень зроблено низку публікацій [10−12], у яких деякі особливості проекту розглянуто докладніше.

ВИСНОВОК

У цій роботі докладно описано комплексний підхід до вирішення одного з актуальних завдань розпізнавання образів – побудови інноваційної системи оптичного розпізнавання текстів на базі сучасних засобів програмування та досягнень у сфері систем комп'ютерного зору. Було сформульовано концепцію побудови системи, що враховує останні досягнення нейромережевих технологій з оптичного розпізнавання символів, що передбачає використання набору алгоритмів підвищення якості роботи системи й навіть широкого спектра сервісних функцій. Попереднє оброблення вихідних даних дає змогу досягти майже 100 % точності розпізнавання тексту.

Необхідно зазначити ще одну важливу мету проекту. У наш час інформаційна безпека стала однією з найважливіших проблем через різке збільшення обсягів даних, що передаються соціальними мережами та хмарними сервісами. Інтернет використовується для передавання великих обсягів даних відкритими мережами та незахищеними каналами, через що може бути порушено конфіденційність особистих і секретних даних, тому необхідно розвивати технології захисту конфіденційних повідомлень. Отже, проведені дослідження актуальні та корисні. Рекомендації щодо спільного використання QR-кодів і методів LSB-стеганографії для кодування та захищеного передавання текстової інформації мають універсальний характер, зокрема, можуть бути використані під час побудови нових і модернізації наявних систем оптичного розпізнавання текстів. Робота має практичну спрямованість. Пропоновані алгоритми реалізовані у вигляді програмних модулів мовою програмування Python і бібліотеки OpenCV. Експериментально досліджено показники якості роботи різних методів кодування та захисту даних. Плануються подальші дослідження інших стеганографічних підходів на основі цифрових зображень, зокрема захист переданих даних із використанням різних показників оцінки включно зі скритністю, візуальною якістю та безпекою.

Мета проєкту – проведення досліджень щодо створення економічної та простої у використанні системи розпізнавання тексту для широкого використання. Програму HQ Scanner можна вважати цілком прийнятним прототипом для вирішення цього завдання.

Отже, цей проєкт є прикладом комплексного підходу до створення систем технічного зору.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. Gradient-Based Learning Applied to Document Recognition / Y. Lecun et al. // Proceedings of the IEEE. − 1988. − vol. 86, no. 11. − P. 2278–2324. doi:10.1109/5.726791.
2. Інформація стосовно бібліотеки ImageNet [Електронний ресурс]. − Режим доступу: https://ua.wikipedia.org/wiki/ImageNet .
3. Krizhevsky, A. ImageNet classification with deep convolutional neural networks / A. Krizhevsky, I. Sutskever, G. Hinton // In Advances in neural information processing systems. – 2012. − P. 1097–1105.
4. ImageNet Classification with Deep Convolutional Neural Networks / A. Krizhevsky et al. // Communications of the ACM. – 2017. − vol. 60, no. 6. − P. 84–90. doi:10.1145/3065386.
5. Програмне забезпечення Caffe [Електронний ресурс]. − Режим доступу: http://caffe.berkeleyvision.org/ .
6. Бібліотека TensorFlow [Електронний ресурс]. − Режим доступу: https://www.tensorflow.org/.
7. Наукова обчислювальна платформа Torch [Електронний ресурс]. − Режим доступу: http://torch.ch/ .
8. Фреймворк нейронної мережі Darknet [Електронний ресурс]. − Режим доступу: https://pjreddie.com/darknet/ .
9. Бібліотека Keras [Електронний ресурс]. − Режим доступу: https://keras.io/.
10. Бібліотека OpenCV DNN [Електронний ресурс]. − Режим доступу: https://github.com/opencv/opencv/tree/master/modules/dnn .
11. Sahu, N. Study on Optical Character Recognition-Techniques / N. Sahy, M. Sonkusare // The International Journal of Computational Science, Information Technology and Control Engineering (IJCSITCE). – 2017. − vol. 4, no. 1. − 14 p. DOI: 10.5121/ijcsitce.2017.4101.
12. Steps Involved in Text Recognition and Recent Research in OCR; A Study / K. Karthick, K. Ravindrakumar, R. Francis, S. Ilankannan // International Journal of Recent Technology and Engineering (IJRTE). – 2019. − vol. 8, iss.1. − P. 3095−3100. − Retrieval Number: A2670058119/19©BEIESP.
13. Програмне забезпечення Adobe Finereader [Електронний ресурс]. − Режим доступу: https://www.abbyy.com/ru/finereader/.
14. Програмне забезпечення OCRopus [Електронний ресурс]. − Режим доступу: https://ua.wikipedia.org/wiki/OCRopus.
15. Програмне забезпечення Tesseract [Електронний ресурс]. − Режим доступу: https://github.com/tesseract-ocr/tesseract.
16. Offline optical character recognition (OCR) method: An effective method for scanned documents / R. Mujibur et al. // 22nd International Conference on Computer and Information Technology (ICCIT). – 2019. − P. 1−15. DOI: 10.1109/ICCIT4885.2019.9038593.

17. Improved OCR quality for smart scanned document management system / Anh Phan Viet et al. // Journal of Science and Technique − Le Quy Don Technical University. – 2020. − no. 210. − P. 51−67.
18. Посібник розробника Python [Електронний ресурс]. − Режим доступу: http://python.org.
19. Підручники з OpenCV − обробка зображень (модуль imgproc) [Електронний ресурс]. − Режим доступу: https://opencv.org/.
20. Python-tesseract − інструмент оптичного розпізнавання символів (OCR) для Python [Електронний ресурс]. − Режим доступу: https://pypi.org/project/pytesseract/.
21. Data pre-processing to increase the quality of optical text recognition systems [Text] / K. Dergachov et al. // Radioelectronic and computer systems. − 2021. – № 4 (100). – С. 183−198. DOI: 10.32620/reks.2021.4.15.
22. Methods and algorithms for protecting information in optical text recognition systems [Text] / K. Dergachov et al. // Radioelectronic and computer systems. − 2022. – № 1 (101). – P. 154−169. DOI: 1032620/reks.2022.1.12.
23. Development of tools for information protection of optical text recognition systems [Text] / K. Dergachov et al. // Radioelectronic and computer systems.− 2022. – № 2 (102). – P. 159−177. DOI: 10.32620/ reks.2022.2.13.
24. A New Approach of Cryptography for Data En-cryption and Decryption [Text] / K. I. Masud et al. // 5th International Conference on Computing and Informatics (ICCI). – 2022. − P. 234-239. DOI: 10.1109/ ICCI54321.2022. 9756078.
25. Assessment of Hybrid Cryptographic Algorithm for Secure Sharing of Textual and Pictorial Content [Text] / P. William et al. // International Conference on Electronics and Renewable Systems (ICEARS). – 2022. – P. 918−922. DOI: 10.1109/ICEARS53579.2022. 9751932.
26. Some Methods of QR code Transmission using Steganography [Text] / D. F. Pastukhov et al. // World of transport and transportation. − 2019. – Vol. 17, Iss. 3. – P. 16–39.
27. QR code image steganography (LSB BIT) with secret image (MSB BIT) using AES cryptography and JPEG compression [Text] / R. Rituraj et al. // Recent Sci-entific Research. − 2019. – Vol. 9, Iss. 7. – P. 27820-27826.
28. Efficiency Assessment of the Steganographic Coding Method with Indirect Integration of Critical Information [Text] / O. Yudin et al. // IEEE International Conference on Advanced Trends in Information Theory (ATIT). – 2019. – P. 36−40. DOI: 10.1109/ATIT49449. 2019.9030473.
29. Li, F. Two-step providing of desired quality in lossy image compression by SPIHT / F. Li, S. Krivenko, V. Lukin // Radioelectronic and computer systems. − 2020. – № 2 (94). – P. 22−32. DOI: 10.32620/reks.2020.2.02.

ЗМІСТ

Навчальне видання

Краснов Леонід Олександрович
Білозерський Владислав Олександрович
Дергачов Костянтин Юрійович
Зимовін Анатолій Якович

ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОЄКТУВАННЯ ТЕХНІЧНИХ СИСТЕМ

Частина 2

СТВОРЕННЯ ТА ВИКОРИСТАННЯ СИСТЕМ ОПТИЧНОГО РОЗПІЗНАВАННЯ ТЕКСТІВ

Редактор А. Г. Литвин

Зв. план, 2024
Підписано до видання 27.04. 2024
Ум. друк. арк. 6,4. Обл.-вид. арк. 7,25. Електронний ресурс

Видавець і виготовлювач
Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»
61070, Харків-70, вул. Чкалова, 17
http://www.khai.edu
Видавничий центр «ХАІ»
61070, Харків-70, вул. Чкалова, 17
izdat@khai.edu

Свідоцтво про внесення суб`єкта видавничої справи
до Державного реєстру видавців, виготовлювачів і розповсюджувачів
видавничої продукції сер. ДК № 391 від 30.03.2001