СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) __Інформаційних технологій____
(повне найменування інституту, факультету) .
__та електроніки_____
Кафедра __Інформаційних технологій та програмування_____
(повна назва кафедри)

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної випускної роботи

освітній ступінь_бакалавр_____
(бакалавр, магістр)
спеціальність _121 Інженерія програмного забезпечення_____
(шифр і назва спеціальності)
спеціалізація _____
(назва спеціалізації)

на тему_Програмне забезпечення перетворення растрових зображень в символьне представлення_____
_____

Виконав: студент групи ІПЗ-20д _____ _А. В. Крохмаль_____
(підпис) (ініціали і прізвище)

Керівник _____ _Д. М. Марченко_____
(підпис) (ініціали і прізвище)

В.о. завідувача кафедри _____ _О. І. Захожай_____
(підпис) (ініціали і прізвище)

Рецензент_____

Київ - 2024

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) __Інформаційних технологій_____
(повне найменування інституту, факультету)
__та електроніки_____
Кафедра __Інформаційних технологій та програмування_____
(повна назва кафедри)
Освітній ступінь __бакалавр_____
(бакалавр, магістр)
спеціальність _121 Інженерія програмного забезпечення_____
(шифр і назва спеціальності)
спеціалізація _____
(назва спеціалізації)

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри
_Захожай О. І._____
“____” _____ 20___року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ ВИПУСКНУ РОБОТУ СТУДЕНТУ

_____Крохмаль Андрій Віталійович_____
(прізвище, ім’я, по батькові)
1. Тема роботи__Програмне забезпечення перетворення растрових зображень_____
__в символьне представлення_____
_____
_____
Керівник роботи_Марченко Дмитро Миколайович, д. т. н., професор_____,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджений наказом університету від “06” травня 2024року №171/15.15-С

2. Строк подання студентом роботи_10.06.2024_____
3. Вихідні дані до роботи_Програмне забезпечення з функціоналом завантаження довільних растрових зображень та формування на їх основі відповідних масивів символів, які візуально відповідають контурам растрового зображення._____
_____
4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)_Вступ, Аналіз методів та засобів обробки та перетворення растрових зображень, Розробка методу перетворення у символьне представлення, Програмна реалізація розробленого методу, Висновки._____
_____
5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслеників)
_____
_____
_____
_____
_____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання__05.02.2024_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання кваліфікаційної випускної роботи	Строк виконання етапів	Примітка
1	Одержання завдання на виконання роботи	05.02.2024	
2	Укладання і погодження з керівником плану і етапів виконання роботи	26.02.2024	
3	Узагальнення даних літературних джерел, аналіз предметної галузі	04.03.2024	
4	Аналіз шляхів виконання завдання. Вибір і погодження з керівником оптимального шляху	25.03.2024	
5	Укладання та тестування програмного продукту	22.04.2024	
6	Укладання, оформлення та погодження пояснювальної записки з керівником	06.04.2024	
7	Здача готової пояснювальної записки на кафедру	03.06.2024	
8	Укладання доповіді і презентації	10.06.2024	

Студент _____ __А. В. Крохмаль_____
(підпис) (ініціали і прізвище)

Керівник роботи _____ __Д. М. Марченко_____
(підпис) (ініціали і прізвище)

ЛИСТ ПОГОДЖЕННЯ І ОЦІНЮВАННЯ

дипломної роботи студента гр. ІПЗ-20д Крохмаль А. В.

Науковий керівник

Проф., д.т.н. _____ Марченко Д. М.

Оцінка наукового керівника: _____

Рецензент Захожай О. І., каф. ІТП (м. Київ), в.о. завідувача кафедри

ПІБ, місто роботи, посада

Оцінка рецензента: _____

Кінцева оцінка за результатами захисту:

_____

Голова ДЕК,

Професор кафедри ІТП

Д.т.н., проф. _____ Меняйленко О.С.

підпис

РЕФЕРАТ

Робота містить: 63 сторінки основного тексту, 8 сторінок додатків, 34 рисунка, 26 використаних джерел.

Метою випускної кваліфікаційної роботи є розробка універсального програмного забезпечення для перетворення растрового зображення у символьне представлення на основі структурного методу з консольним та графічним інтерфейсом користувача.

У ході проведеного дослідження було проаналізовано існуючі методи та засоби попередньої обробки растрових зображень та перетворення зображень у символьне представлення. Обґрунтовано використання методу перетворення за принципом співставлення структури. Розроблено метод перетворення растрового зображення у символьне представлення за структурним принципом, визначено його переваги та недоліки. Розроблено програмне забезпечення з консольним та графічним інтерфейсом користувача, що реалізує запропонований метод.

Програмне забезпечення реалізоване відповідно до всіх вимог технічного завдання. Зроблено детальний опис процесу розробки, а також продемонстрована робота готових програмних продуктів.

Ключові слова: РАСТРОВЕ ЗОБРАЖЕННЯ, СИМВОЛЬНЕ ПЕРЕТВОРЕННЯ ЗОБРАЖЕНЬ, ІНФОРМАТИВНІСТЬ ДАНИХ, ЗГОРТКОВА НЕЙРОННА МЕРЕЖА, ОБРОБКА РАСТРОВОГО ЗОБРАЖЕННЯ

ЗМІСТ

ВСТУП

Актуальність дослідження. Перетворення растрового зображення до символьної форми є не тільки формою образотворчого мистецтва, що набуло популярності ще у 1980-х роках, а й формою попередньої обробки і трансформації графічних даних з метою мінімізації займаного об’єму пам’яті та селекції інформативних ознак. В ньому для представлення візуальної інформації замість пікселів растрового зображення використовуються символи. При створенні такого зображення використовується палітра, що складається з буквених, цифрових символів та символів знаків пунктуації з таблиці ASCII [1, 2].

Створення символьного представлення зображення може відбуватися в декілька способів: використанням спеціалізованих графічних редакторів для ASCII- графіки, програмного забезпечення для рендерингу тексту чи векторної графіки символами ASCII, перетворення растрових зображень за допомогою спеціалізованих конверторів.

Багато програм, які існують в даний момент для перетворення растрових зображень у символьні рисунки, мають вузьку спеціалізацію та обмежену підтримку різних платформ. Більшість з них не здатні ефективно вирішувати завдання перетворення зображень для публікації в Інтернеті, соціальних мережах тощо [3].

Тому, створення універсального програмного забезпечення, яке здатне ефективно перетворювати растрові зображення в символьні рисунки для широкого спектру завдань, є актуальною науково-технічною задачею.

Об’єкт дослідження – технологія перетворення растрового зображення у символьне представлення.

Предмет дослідження – структурний метод символьного перетворення.

Мета дослідження – розробка універсального програмного забезпечення для перетворення растрового зображення у символьне

представлення на основі структурного методу з консольним та графічним інтерфейсом користувача.

Завдання дослідження:

а) аналіз існуючих методів обробки растрових зображень;
б) аналіз методів та засобів перетворення зображень у символьне представлення;
в) обґрунтування актуальності використання структурного методу перетворення;
г) розробка методу перетворення растрового зображення у символьне представлення за структурним принципом;
д) реалізація розробленого методу в програмному забезпеченні.

Методологічна та теоретична основа дослідження – методи комп'ютерного зору: бінаризації, фільтрації, сегментації, методи машинного навчання: використання каскадів Хаара, методи глибокого навчання.

Методи дослідження – загальнонаукові: аналіз і синтез, порівняння, експеримент; конкретнонаукові: метод адаптивної гаусівської бінаризації, метод двосторонньої фільтрації, метод морфологічного стоншення, метод ковзного вікна, використання згорткової нейронної мережі.

Практичне значення отриманих результатів. Розроблене програмне забезпечення може бути використано в широкому спектрі задач, включаючи застосування в художній практиці, поширення перетворених зображень в мережі Інтернет, використання в системах розпізнавання образів (СРО) для виділення інформативних ознак об'єктів.

1. АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ ОБРОБКИ ТА ПЕРЕТВОРЕННЯ РАСТРОВИХ ЗОБРАЖЕНЬ

1.1 Попередня обробка растрових зображень

Існує широке різноманіття методів попередньої обробки растрових зображень, які можна класифікувати наступним чином:

– методи, основані на фільтрації (бінаризація, підкреслення контурів, видалення фону, формування півтонових моделей, сегментація тощо);

– методи, основані на формуванні векторної моделі растрового зображення.

Методи, основані на фільтрації, хоча і дозволяють певним чином спростити растрове зображення, але не виключають більшості неінформативних аспектів, а розмірність інформаційного поля залишається відповідним до роздільної здатності матриці растрового зображення. Крім цього, формат растрового зображення для достовірного співставлення вимагає вирішення додаткових питань компенсації зсуву, обертання, афінного перетворення тощо.

Методи, основані на векторизації растрового зображення, також не дають однозначно достовірного результату, так як вимагають вирішення додаткових питань товщини ліній, компенсації чи додаткового перетворення ліній, що накладаються одна на іншу. Також така геометрична модель співставлення буде містити велику кількість неінформативних та дубльованих даних [4].

Перетворення кольорового зображення у відтінки сірого кольору дозволяє виділити із зображення інформацію про яскравість пікселів та видалити інформацію про кольори. Він використовується для зменшення розмірності даних та, в підсумку, дозволяє знизити обчислювальне навантаження на систему розпізнавання.

Наприклад, перетворення зображення з формату ARGB8888 (32 bpp) у відтінки сірого (8 bpp) дозволить зменшити розмірність даних, що

обробляються в 4 рази. Щоправда використовувати даний метод слід у випадках, коли колір предмета на зображенні неважливий.

Методи бінаризації дозволяють зменшити розмірність даних за рахунок зниження розрядності кодування кожного пікселя зображення. Бінарна форма дозволяє значно зменшити обсяги даних, що підлягають зберіганню та обробці. Наприклад, один піксель зображення у форматі ARGB8888 займатиме 32 біта, у той час як його бінарний еквівалент – 1 біт. Бінаризація за пороговим значенням є одним з найпростіших алгоритмів обробки зображення, коли усі пікселі зображення, атрибут яких перевищує апріорно задане порогове значення, кодуються бітом 1 (білий колір), а інші – бітом 0 (чорний колір). Результат порогової бінаризації наведено на рисунку 1.1.1 (б).

Очевидно, що при бінаризації за пороговим значенням не завжди існує поріг, за якого би не відбувалась втрата інформативних даних. Крім того, результат даного методу досить сильно залежить від освітленості сцени спостереження об’єкта розпізнавання. Частково цей недолік можна виправити, застосувавши ітераційний метод Оцу [5], однак при нерівномірній освітленості зображення результат все одно буде незадовільним. Тому виникає необхідність у адаптивній зміні порогового значення на кожній окремій ділянці зображення (зазначення локального порогу) [6]. Такий підхід реалізовано в методах адаптивної бінаризації. Один з варіантів реалізації даного методу – розбиття зображення на сітку комірок і застосування до кожної з них методу порогової бінаризації. Інший варіант – встановлення порогового значення для кожного пікселя зображення на основі дослідження його деякого оточення. Локальний поріг у такому випадку розраховується на основі середнього арифметичного значення яскравості пікселів околу або їхньої гаусівської суми. Результат роботи останнього варіанту наведено на рисунку 1.1.1 (в), для якого можна зазначити меньший рівень втрат інформативних ознак [7].

а) б) в)

Рисунок 1.1.1 – Варіанти бінаризації зображення: а) початкове зображення; б) порогова бінаризація; в) адаптивна бінаризація

Виділення контурів використовується для зображень, де контури об’єктів на сцені спостереження є найбільш інформативними в контексті поставленої задачі. Цей метод дає більш чітке представлення про контури об’єкта на сцені, ніж попередній. Одним із варіантів реалізації даного методу є алгоритм Кенні [8]. Сутність цього алгоритму полягає у пошуку градієнтів та знаходженні їх локальних максимумів. Результат роботи такого алгоритму проілюстровано на рисунку 1.1.2. Однак, таке рішення не є універсальним, так як разом с підкресленням контурів на зображенні втрачаються інформативні ознаки щодо тла об’єкта, текстур та поверхонь [7].

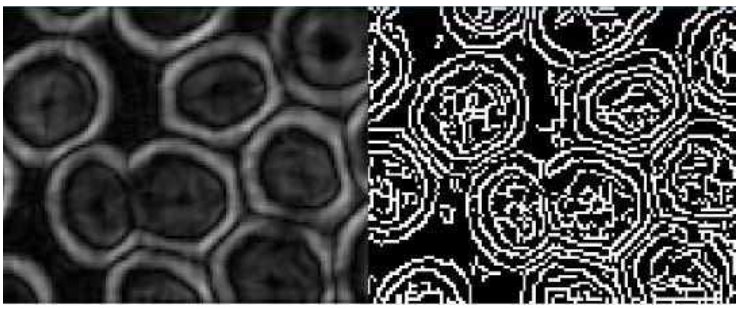


Рисунок 1.1.2 – Виділення контурів об’єктів на сцені спостереження

Метод векторизації аналогічний методу пошуку контурів. Відмінністю є збереження контурів у векторному представленні. В деяких задачах це полегшує роботу з ними, а також значно зменшує об’єм оброблюваних даних у випадку оперування з нескладними формами об’єкту

на растровому зображенні високої роздільної здатності. Але зазначені недоліки попереднього методу також притаманні цьому [7].

Фільтрація є одним із основних методів обробки зображення для видалення неінформативних даних зображення, таких як шум, текстура, дрібні деталі тощо. В залежності від поставленої задачі використовується певний вид фільтрації. Існують кольорові фільтри, такі як виділення RGB-каналів, виділення кольорового діапазону, пошук максимальної кольорової компоненти тощо. Даний тип фільтрів застосовується в задачах, де інформативним є саме колір. Недоліком методів фільтрації є чутливість до зміни кольору, що не є бажаним при використанні зображень, отриманих різними технічними засобами реєстрації за різних умов спостереження об’єкту розпізнавання [7]. Приклад фільтрації кольорового діапазону наведено на рисунку 1.1.3.

Рисунок 1.1.3 – Фільтрація кольорового діапазону

Інший тип фільтрації не є чутливим до кольору та використовується в першу чергу для видалення шуму. Роздивимося найважливіші з них. Фільтрація за Гаусом: використовується для розмиття всього зображення із розмиттям контурів об’єктів на ньому. Типове застосування – видалення цифрового шуму чи артефактів стиснення, дрібних деталей зображення.

Двосторонній фільтр або подвійна фільтрація за Гаусом: використовується для розмиття однорідних ділянок зображення. На відміну від попереднього, даний метод враховує різницю в яскравості у кольорі пікселів зображення, внаслідок чого отримане зображення зберігає границі та

контури об’єктів. Двосторонній фільтр дозволяє видалити шуми, текстури, дрібні дефекти тощо, залишаючи чіткими контури предметів.

Медіанний фільтр: використовується для видалення «шуму солі та перцю» [9], тобто контрастних білих точок на чорному тлі чи чорних точок на білому. Сутність даного методу полягає у проході по зображенню матрицею пікселів розміром 3x3 чи 5x5. Значення середнього пікселя розраховується як медіана значень інших пікселів у матриці. Цей метод також зберігає ясність контурів, що робить його зручним [7]. Приклади зображено на рисунку 1.1.4.

а) б) в) г)

Рисунок 1.1.4 – Варіанти фільтрації зображення: а) початкове зображення; б) фільтрація за Гаусом; в) медіанний фільтр; г) двосторонній фільтр

Морфологічні перетворення – операції над бінарними зображеннями, що призводять до зміни форми його структурних частин. Основними морфологічними операціями є диляція та ерозія, а також їхні комбінації – відкриття та закриття [10].

Операція диляції розширює та стовщує область зображення, ерозія – навпаки стоншує. Результат цих операцій наведений на рисунку 1.1.5. Слід зазначити, що такі морфологічні перетворення дозволяють певним образом підкреслити інформативні ознаки, але значною мірою не впливають на обсяги даних. Операція відкриття є диляцією ерозії. Її використовують для видалення дрібних об’єктів із зображення при збереженні форми великих об’єктів. Операція закриття є ерозією диляції. Використовувати дану операцію можна

для видалення дрібних отворів на зображенні. Розміри та форма великих об’єктів при даній операції також залишаються незмінними. Результати операцій відкриття та закриття наведені на рисунку 1.1.6.

а) б) в)

Рисунок 1.1.5 – Приклади морфологічних перетворень: а) початкове зображення; б) диляція; в) ерозія

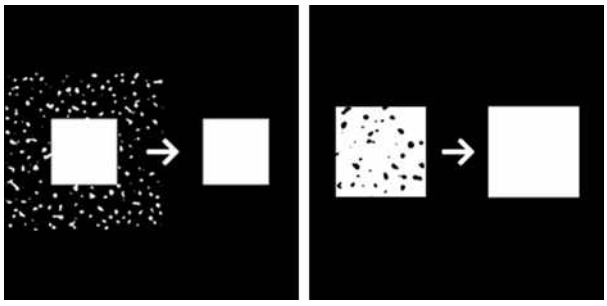
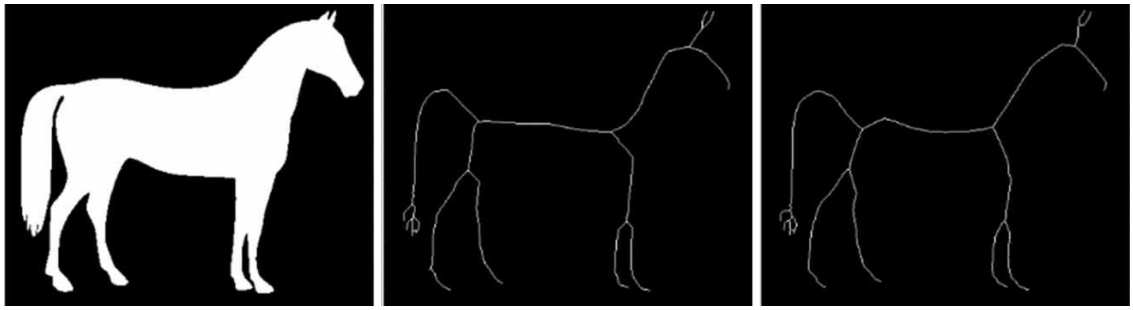


Рисунок 1.1.6 – Операції відкриття та закриття

Скелетонізація – це морфологічна операція, що стоншує всі ділянки зображення до товщини в один піксель по середнім лініям. Стоншення – операція аналогічна скелетонізації, виконується ітераціями, зменшуючи товщину ліній зображення до товщини в один піксель. Відмінністю скелетонізації є те, що вона зберігає розміри оригінального зображення: лінії, що утворилися, обов’язково досягають границі початкового зображення. Тому стоншення дає більш гладкий контур, який нагадує форму початкового зображення [7]. Результати операцій скелетонізації та стоншення наведені на рисунку 1.1.7. Недоліки такого методу перетворення растрових зображень аналогічні виділенню контурів.



а) б) в)

Рисунок 1.1.7 – Операції скелетонізації та стоншення: а) початкове зображення; б) склетонізація; в) стоншення

Сегментація – це поділ зображення на сегменти, що відповідають об’єктам різних класів приналежності.

Існує два типи сегментації зображень: семантична та екземплярна. Результатом семантичної сегментації є множина сегментів, кожен з яких є зображенням усіх предметів певного класу. Результатом екземплярної сегментації, на відміну від попереднього типу, є множина сегментів, які є зображеннями кожного предмета певного класу приналежності [11]. Обидва типи сегментації широко застосовуються залежно від предметної області. Головною перевагою сегментації зображення є виділення тільки досліджуваного об’єкта та видалення фону поза ним. Даний метод використовується, коли інформативним є об’єкт, а фон не є потрібним для розпізнавання [7]. Результати обох типів сегментації зображення наведено на рисунку 1.1.8.

а) б) в)

Рисунок 1.1.8 – Семантична та екземплярна сегментація: а) початкове зображення; б) семантична сегментація; в) екземплярна сегментація

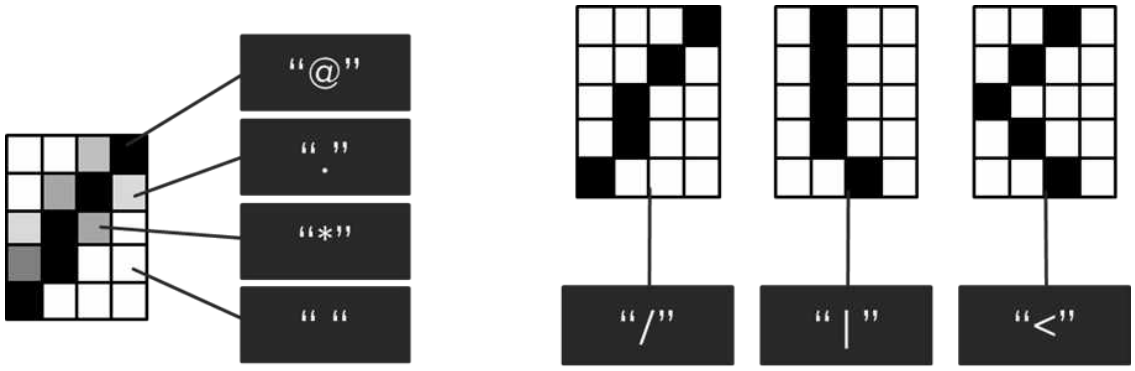
Зазвичай, для сегментації зображення необхідна додаткова система розпізнавання. Тому актуальність застосування даного методу в тій чи іншій задачі залежить від швидкодії процесу обробки даних. Окрім того, в процесі як семантичної так і екземплярної сегментації видаляються дані фону, які в певних прикладних застосуваннях можуть бути інформативними. Крім цього, сегментовані елементи зображення мають також растровий формат, який вимагає, відповідно, попіксельного збереження даних та не дозволяє значно економити на обсягах пам’яті [7].

Використання штучних нейронних мереж. До даної категорії можна віднести будь-які методи обробки зображень, в яких застосовуються штучні нейронні мережі (ШНМ). Використання останніх зазвичай обумовлюється складністю чи неможливістю алгоритмічної реалізації певного методу обробки. Прикладами використання ШНМ в обробці растрових зображень є автоматична обрізка, сегментація, видалення фону за об’єктом, зміна представлення на векторне, символьне тощо.

1.2 Перетворення у символьне представлення

На даний момент існує велика кількість методів конвертації растрового зображення в символьний малюнок. Їх можна класифікувати за ознакою, що визначає співвідношення між растровою матрицею та матрицею символів ASCII. Можна виділити методи перетворення зображень в малюнок

символами, що використовують як ознаку розділення тон та структуру [2, 3, 7, 12] (рисунок 1.2.1).



а) б)

Рисунок 1.2.1 – Методи перетворення у символьне представлення на основі співставлення: а) яскравості пікселів; б) структури підматриці пікселів

Методи на основі співставлення тону визначають співвідношення між зображенням та символьною матрицею, використовуючи тон (колір чи середню яскравість) пікселя. Зазвичай, в цьому методі одному пікселю відповідає один символ, що обирається з таблиці пошуку за принципом співвідношення чорного та білого кольорів в зображенні цього символу (рисунок 1.2.1 (а)). Наприклад, градієнт від білого до чорного можна утворити символами « », «.», «-», «*», «@» чи іншими. В даному методі також можуть бути використані символи псевдографіки, що позначають заливку (такі як «▒»), для відображення пікселів різної яскравості. При цьому можливий варіант використання як чорних символів на білому тлі, так і білих на чорному. Приклади символьних малюнків, створених із зображень за даним методом, наведені на рисунку 1.2.2.

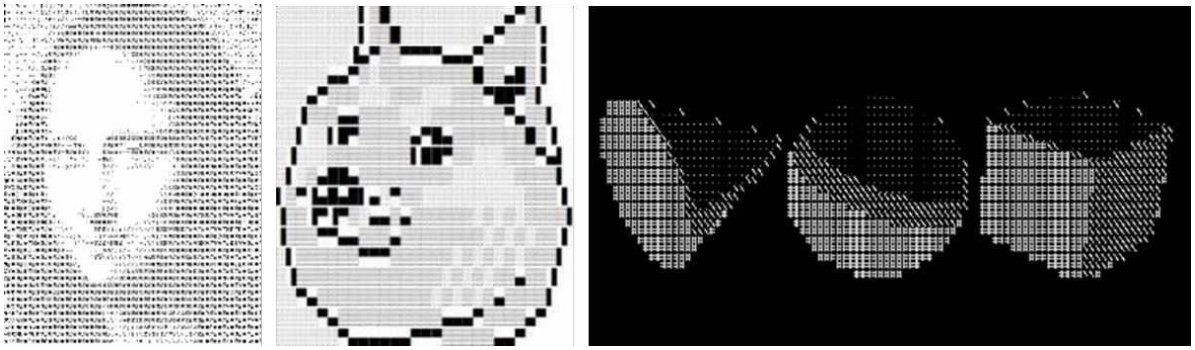


Рисунок 1.2.2 – Приклади символьних зображень, створених за принципом співставлення тону

Зазвичай, для реалізації метода використовують алгоритми інтерполяції та дизерингу. Результат даного метода аналогічний із методом переведення зображення у відтінки сірого. Розмірність даних також аналогічна (при співставленні одного пікселя одному символу та використанні набору символів ASCII) і становить 8 біт на піксель. Тому використання даного метода в СРО не є прийнятним [7].

Методи на основі співставлення структури ґрунтуються на співставленні структури виділеної підматриці матриці пікселів растрового зображення символам таблиці ASCII [13]. Зазвичай, декільком пікселям, що входять до підматриці, відповідає один символ ASCII. Він визначається за схожістю форми з зображенням підматриці пікселів (рисунок 1.2.1 (б)). Наприклад, символами «_», «/», «\», «|» позначаються лінії контурів, направлені під різними кутами. Приклади зображень, створені за структурним методом, наведені на рисунку 1.2.3.

Рисунок 1.2.3 – Приклади символьних зображень, створених за принципом співставлення структури

Для реалізації цього методу використовують алгоритми пошуку найкращого співпадіння або ШНМ. З метою спрощення алгоритмів конвертації перетворення краще робити над бінарним зображенням. Створені структурним методом малюнки є меншими за розмірами та об’ємом займаної пам’яті та мають виразні контури, через що краще сприймаються візуально. Коефіцієнт стиснення при використанні даного метода [7] (при використанні набору символів ASCII) можна обчислити за формулою:

$$K = \frac{a*b}{8} * 100\%,$$

де a і b – розміри підматриці відповідного бінарного зображення для співставлення.

1.3 Аналіз існуючого програмного забезпечення

ASCII Art Maker

Даний інструмент дозволяє створювати ASCII-графіку з тексту та зображень. Працює з багатьма популярними форматами растрової графіки. Підтримується багато форматів вихідних даних, серед яких HTML, TXT, RTF та деякі растрові формати. Програма використовує моноширинні шрифти. В основі лежить принцип співставлення яскравості пікселів символам. Є можливість налаштування параметрів попередньої обробки, таких як яскравість, контрастність тощо. Існує функція ручного редагування отриманої символьної послідовності. Програма проста у використанні та має меню довідки [14]. Програмне забезпечення безкоштовне та знаходиться у вільному доступі. Підтримується операційними системами MS Windows. Зовнішній вигляд інтерфейсу наведено на рисунку 1.3.1.

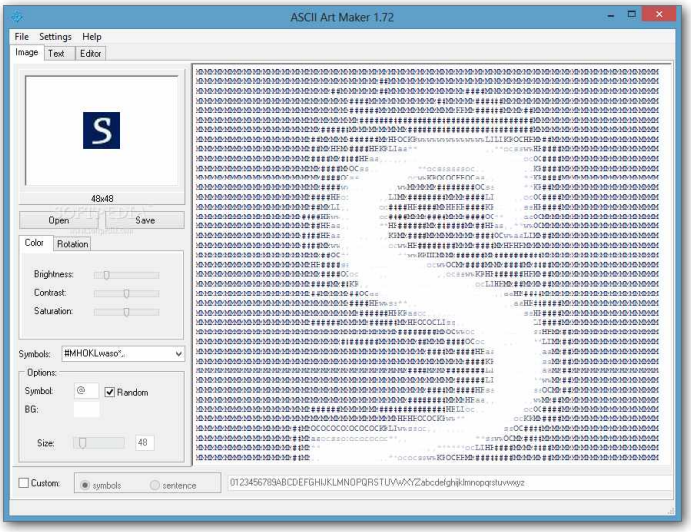


Рисунок 1.3.1 – Вигляд інтерфейсу програми ASCII Art Maker

Characterizer

Даний інструмент перетворює растрові зображення на символьну графіку. Працює з форматами растрових зображень JPG, BMP та WMF. Вихідним форматом є текстовий документ з розширенням *.TXT. В основі лежить принцип співставлення яскравості пікселів символам. Для відображення тексту використовуються моноширинні шрифти. Програма використовує алгоритми попередньої обробки з налаштуванням селекції інформативних ознак, таких як контур чи текстура [15]. Програмне забезпечення безкоштовне та знаходиться у вільному доступі. Підтримується операційними системами MS Windows. Зовнішній вигляд інтерфейсу наведено на рисунку 1.3.2.

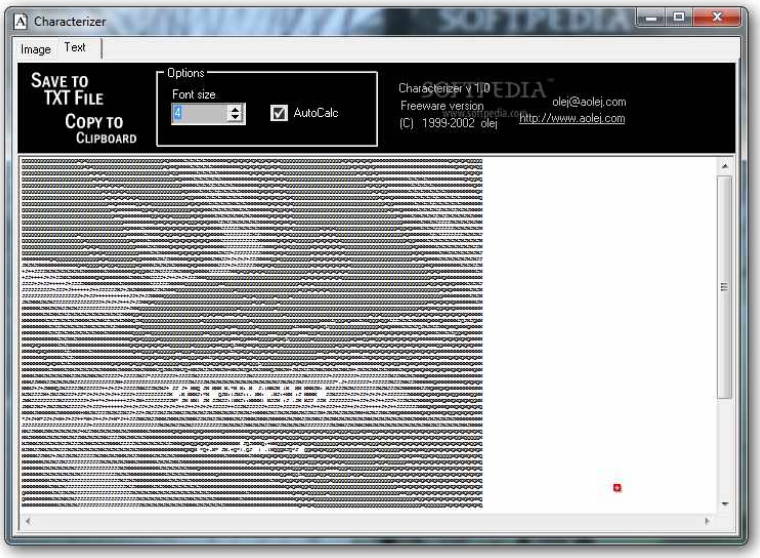


Рисунок 1.3.2 – Вигляд інтерфейсу програми Characterizer

ImageToText

Дане програмне забезпечення є інструментом, що конвертує зображення в символи ASCII. Підтримується багато форматів вхідних даних, серед яких JPG, JPEG, BMP, GIF, PNG, TIF, TIFF, ICO, EMF, WMF. Програма має можливість збереження вихідних даних у цих растрових форматах, а також може конвертувати зображення в HTML, RTF чи TXT файли. Передбачена можливість пакетної обробки декількох зображень одночасно. Формат вихідного тексту налаштовується користувачем. Передбачено налаштування розміру шрифту, алфавіту символів, тонке налаштування кольорів [16]. Інструмент підходить як для початківців, так і для професіоналів. Програма є умовно-безкоштовною та пропонує безкоштовну пробну версію. Ціна повної версії складає $5.95. Підтримується операційними системами MS Windows. Зовнішній вигляд інтерфейсу наведено на рисунку 1.3.3

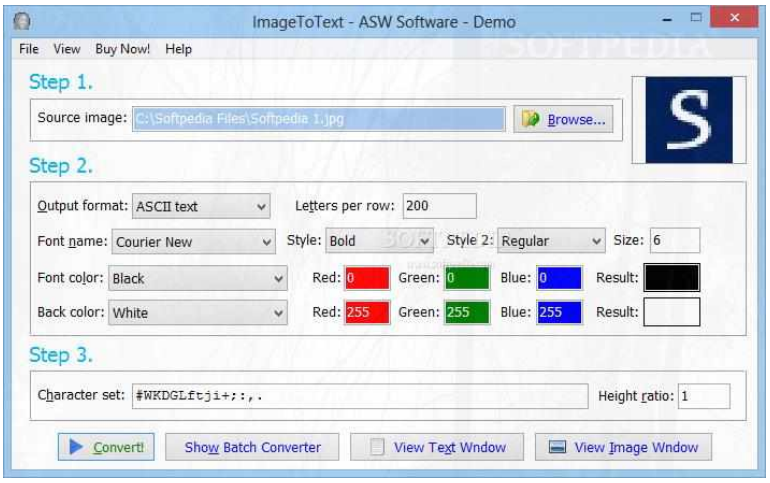


Рисунок 1.3.3 – Вигляд інтерфейсу програми ImageToText

ASCII Generator 2

Даний конвертер зображень у текстову графіку працює з багатьма популярними форматами растрових зображень. Вихідним форматом є текстовий документ з неформатованим текстом, rich text, NFO або XHTML. Також є можливість збереження зображення, утвореного рендерингом згенерованого тексту в растровому форматі JPEG, BMP, GIF, PNG або TIF. В основі лежить принцип співставлення яскравості пікселів символам. Для відображення тексту використовуються моноширинні шрифти. Є можливість налаштування параметрів попередньої обробки, таких як яскравість, контрастність, дизеринг тощо. Передбачено налаштування алфавіту символів шляхом вибору одного із запропонованих наборів. Програма дозволяє генерувати зображення білими знаками на чорному фоні, чорними знаками на білому фоні, а також кольорових зображень. Однак є деякі обмеження, такі як відсутність кнопки скасування та складність попереднього перегляду зображення [17]. Програмне забезпечення безкоштовне, має відкритий вихідний код та знаходиться у вільному доступі. Воно не потребує інсталяції та може запускатися з зовнішнього пристрою. Підтримується операційними системами MS Windows. Зовнішній вигляд інтерфейсу наведено на рисунку 1.3.4.

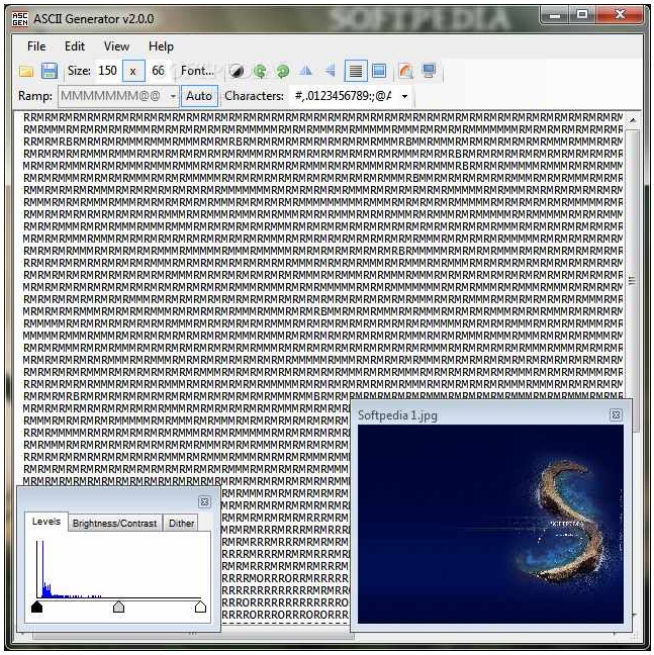


Рисунок 1.3.4 – Вигляд інтерфейсу програми ASCII Generator 2

Asciiart.club

Це онлайн-генератор ASCII-арту. Працює з багатьма популярними форматами растрових зображень. Вихідним форматом є створена символьна послідовність або зображення, утворене шляхом її рендерингу в растровому форматі PNG. В основі лежить принцип співставлення яскравості пікселів символам. Для відображення тексту використовуються шрифти як фіксованої, так і змінної ширини символів. Є можливість налаштування яскравості та порогових значень чорного та білого кольору. Передбачено налаштування алфавіту символів. Вбудовані набори включають як стандартні символи ascii, так і знаки псевдографіки, символи грецького та японського алфавітів. Програма має функціонал для генерації кольорових зображень. Є обмеження, пов’язані з попереднім переглядом зображення [18]. Програмне забезпечення безкоштовне та знаходиться у вільному доступі. На сайті також представлена галерея створених користувачами символьних малюнків. Зовнішній вигляд інтерфейсу наведено на рисунку 1.3.5.

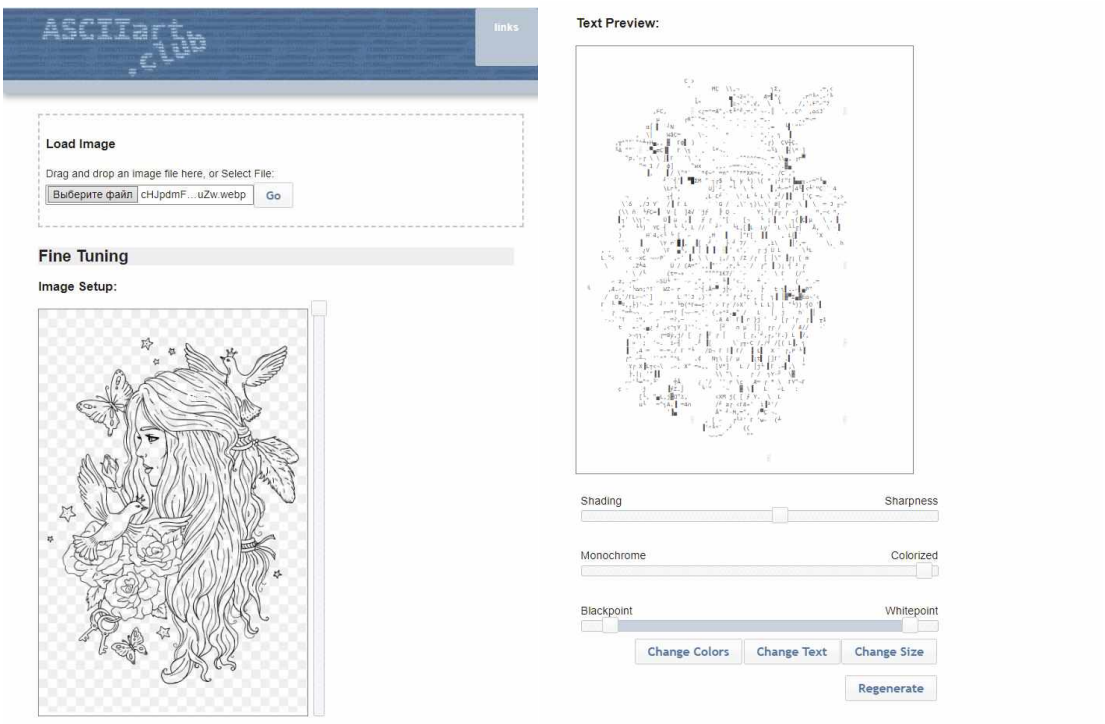


Рисунок 1.3.5 – Вигляд інтерфейсу web-застосунку asciiart.club

DeepAAonWeb

Це веб-інструмент для обробки зображень. Працює з багатьма популярними форматами растрових зображень. Вхідними даними є зображення, виконане чорною лінією товщиною в 1 піксель на білому фоні. Вихідним форматом є створена символьна послідовність. В основі лежить принцип співставлення структури матриці пікселів ковзного вікна символам. Для відображення тексту використовуються шрифт змінної ширини. Є можливість налаштування розмірів вхідного зображення. Обробка виконується послідовно методом ковзного вікна. Поточне положення вікна та результат співставлення відображаються в реальному часі. Веб-застосунок попереджає користувачів, що він працює лише в браузері Google Chrome, а завантажене зображення буде зберігатися локально. Програмне забезпечення безкоштовне та знаходиться у вільному доступі. Зовнішній вигляд інтерфейсу наведено на рисунку 1.3.6.

Рисунок 1.3.6 – Вигляд інтерфейсу web-застосунку DeepAAonWeb

В результаті аналізу існуючих варіантів програмного забезпечення для перетворення растрового зображення у символьне представлення можна зробити наступні висновки:

– Переважна більшість існуючого програмного забезпечення ґрунтується на групі методів, що базуються на співставленні яскравості. Програми, що використовують структурні методи, розповсюджені слабо.

– Деякі конвертори, в основі яких лежать методи другої групи, потребують в якості вхідних даних монохромний рисунок, що створює проблеми для обробки реальних фотографій.

– Багато подібного програмного забезпечення не є кросплатформеним та має проблеми з програмною сумісністю. Також відсутня підтримка різних апаратних засобів для оптимізації швидкості обчислень.

– Більшість програм є вузькоспеціалізованими, спектр задач використання в системах розпізнавання образів взагалі не охоплений.

Тому в дипломній роботі було поставлено задачу розробки програмного забезпечення для перетворення будь-якого (в тому числі фотографічного) зображення в символьний рисунок за структурним методом.

2. РОЗРОБКА МЕТОДУ ПЕРЕТВОРЕННЯ У СИМВОЛЬНЕ ПРЕДСТАВЛЕННЯ

Для розробки програмного забезпечення було розроблено модифікований метод структурної трансформації растрового зображення, особливістю якого є виділення інформативних даних за допомогою автоматичної обрізки та видалення фону, використання алгоритмів фільтрації, бінаризації, морфологічного стоншення контурів [1-4, 7]. Перетворення зображення в символьну послідовність здійснюється глибокою ШНМ.

2.1 Вимоги та граничні умови до реалізації методу

– Вхідними даними є растрове фотографічне зображення об’єкта (людини) у довільній обстановці.

– Вихідними даними є текстовий документ із отриманим зображенням у символьному представленні, що є зручним для візуального сприйняття людиною.

– На етапі виділення досліджуваного об’єкта на фото інформативними даними вважатиметься зображення об’єкта (людини або її обличчя). Неінформативними даними на цьому етапі є оточуюча людину обстановка (фон) та інші об’єкти на фото окрім людини.

– Інформативними даними на етапі перетворення зображення об’єкта в символьне представлення є його контури. Неінформативними на даному етапі будуть колір та текстура.

– Розмірність даних повинна бути мінімальною завдяки перетворенню зображення у символьне представлення за принципом співставлення структури.

– Якість виділення контурів має мінімально залежати від освітленості та контрастності зображення.

– Час на обробку зображення повинен бути мінімізований шляхом обрізки зображення перед його подальшою обробкою (вхідне зображення повинно бути обрізане до розмірів об’єкта на ньому).

– Метод повинен бути зручним для ручного налаштування деяких параметрів обробки, таких як константи фільтрації, порогове значення бінаризації чи необхідність обрізки.

– Метод має бути зручним для переробки під задану предметну область.

2.2 Структура методу перетворення

У розробленому методі обробка зображення відбувається в 6 стадій. Структурна схема перетворень та зміни зображення на кожній стадії перетворення наведено на рисунку 2.2.1.

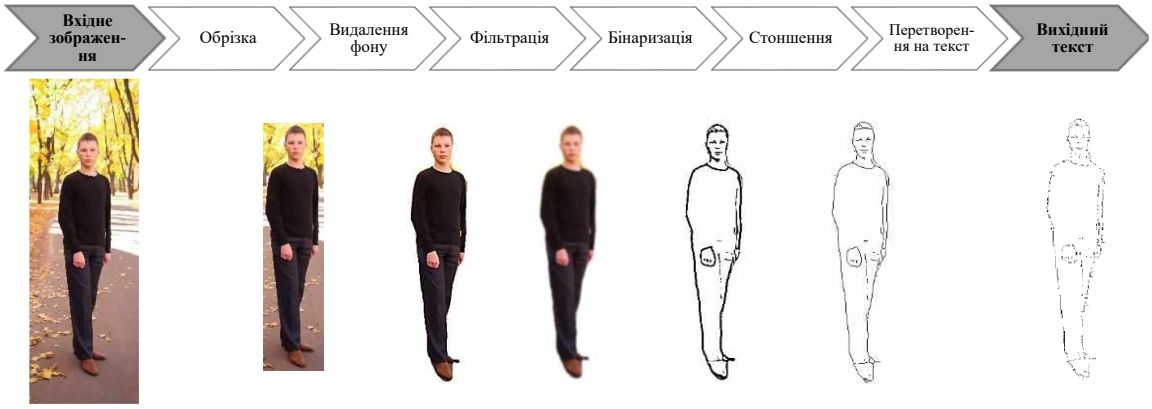


Рисунок 2.2.1 – Зображення на кожному з етапів обробки

Розглянемо докладніше кожну стадію перетворення окремо.

Обрізка. На даному етапі вхідне зображення обрізається до розмірів об’єкту (людини чи її обличчя). Це дозволяє, по-перше, видалити із зображення непотрібні об’єкти та залишити тільки той, який досліджується, а по-друге, зменшити розміри зображення для подальшої обробки з метою зменшення обчислювального навантаження. Для можливості гнучкого налаштування дана стадія може бути пропущена, виконана автоматично чи виконана вручну. Перший варіант може бути застосований користувачем в ситуації, коли вхідне зображення вже обрізане до розмірів досліджуваного об’єкта. Також може бути здійснена ручна обрізка через інтерактивний

графічний інтерфейс. Іншим варіантом є автоматична обрізка, яка відбувається без участі людини. Її сутність полягає у розпізнаванні досліджуваного об’єкта на зображенні та обрізці останнього по координатах області, в якій цей об’єкт знаходиться. У запропонованому методі було використано класифікатор на основі каскадів Хаара [20] для визначення людини чи людського обличчя. Однак даний метод класифікації відзначається невисокою точністю у визначенні людини. Для запобігання цьому може бути застосований класифікатор на основі гістограм орієнтованих градієнтів, який показує більшу точність в даній задачі. Вхідне зображення та результати автоматичної обрізки обличчя людини наведені на рисунку 2.2.2.

Рисунок 2.2.2 – Режими автоматичної обрізки зображення

Видалення фону. Дана стадія може бути як пропущеною за умови, що фон на зображенні вже видалено чи він є однорідним, так і виконано вручну чи автоматично. Ручне видалення фону відбувається за допомогою інтерактивного інтерфейсу, який пропонує користувачу відмітити ділянки зображення для залишення та для видалення. Також додано можливість автоматичного видалення фону без участі людини. Це реалізовано на основі ШНМ для семантичної сегментації зображень [21]. Таким чином з фотографії вирізаються лише сегменти, що відповідають зображенням об’єктів (людей) на ній. Попередній етап (обрізка) дозволяє максимально зменшити обчислювальне навантаження на етапі сегментації. Альтернативою семантичної сегментації є використання ШНМ для пошуку об’єктів на передньому плані. Відмінністю у методах є те, що останній дозволяє вирізати

будь-який об’єкт переднього плану незалежно від його класу приналежності. Коли фон видалено, зображення додатково обрізається до розмірів об’єкта для видалення зайвих областей, що відповідають фону. Після цих двох стадій отримується зображення з максимально зменшеною розмірністю, до якого будуть застосовані наступні перетворення.

Фільтрація за допомогою двостороннього фільтра. На цій стадії зображення піддається фільтрації для видалення дрібних об’єктів, текстур, цифрового шуму, артефактів стиснення з метою запобігання надлишковості даних у вихідному зображенні. Двосторонній фільтр був обраний через його властивість залишати контури предметів на зображенні чіткими і згладжувати лише дрібні деталі. Двосторонній фільтр або подвійна фільтрація за Гаусом використовується для розмиття однорідних ділянок зображення. Даний метод фільтрації враховує різницю в яскравості та кольорі пікселів зображення, внаслідок чого отримане зображення зберігає границі та контури об’єктів. Двосторонній фільтр дозволяє видалити шуми, текстури, дрібні дефекти тощо, залишаючи чіткими контури предметів. Чіткість контурів є необхідною умовою для коректної роботи наступної ланки ланцюга перетворень – гаусівської бінаризації.

Бінаризація з адаптивним пороговим значенням за Гаусом. На наступному етапі необхідно зменшити розрядність кольору зображення до 1 біту на піксель, зважаючи на необхідність збереження контурів зображення та видалення інформації про кольори частин об’єкта. Отже, для досягнення цієї мети було обрано метод адаптивної бінаризації за Гаусом. Він використовує встановлення порогового значення для кожного пікселя зображення на основі дослідження його деякого околу. Локальний поріг бінаризації у такому випадку розраховується на основі гаусівської суми яскравостей пікселів околу. Даний метод дозволяє бінаризувати зображення зі збереженням контурів та видаленням кольору в їх середині. Також метод показує себе з гарної сторони при різних рівнях освітленості та контрастності об’єктів. Також можливе тонке налаштування бінаризації шляхом підбору відповідних констант.

Стоншення темних ділянок до товщини в 1 піксель. Дана стадія необхідна для коректного функціонування наступного етапу перетворення (переходу в символьне представлення). Результатом гаусівської бінаризації є контури неоднакової товщини, яку необхідно вирівняти для спрощення процесу перетворення у символьне представлення. Для цього було використано морфологічну операцію стоншення, так як на відміну від скелетонізації, вона дає лінії, найбільш схожі на контури вхідного зображення. Скелетонізація – це морфологічна операція, що стоншує всі ділянки зображення до товщини в один піксель по середнім лініям. Стоншення – операція аналогічна скелетонізації, виконується ітераціями, зменшуючи товщину ліній зображення до товщини в один піксель. Відмінністю скелетонізації є те, що вона зберігає розміри оригінального зображення: лінії, що утворилися, обов'язково досягають границі початкового зображення. Тому стоншення дає більш гладкий контур, який нагадує форму початкового зображення. Традиційно у морфологічних операціях зображенням вважаються білі ділянки, а фоном чорні, тому в запропонованому методі зображення інвертується двічі: перед та після перетворення. В результаті отримується зображення чорними лініями товщиною в 1 піксель на білому фоні. Ця стадія була останньою з тих, що на виході дає растрове зображення. Залишається лише власне здійснити перехід від растрового до символьного представлення зображення.

Перетворення у символьне представлення. Цей етап є ключовим у даному методі. На вході мається бінарне зображення з товщиною ліній в 1 піксель, на виході отримується масив строкових даних, що є символьним представленням вхідного зображення. Перетворення відбуватиметься за структурним принципом. Сутність підходу полягає у проході по зображенню вікном 64x64 пікселі та співставленні структури підматриці пікселів центральної області вікна розміром 16x16 пікселів одному з 411 символами кодової таблиці Shift JIS (рисунок 2.2.3). Тобто центральна частина вікна відповідає поточній ділянці зображення, для якої відбувається перетворення

на відповідний символ. Ділянка вікна навкруги його центральної області відповідає за врахування контексту. На розглянутому прикладі, вигляд символів відповідає гліфам шрифту 12-pt MS PGothic з 2-піксельними пробілами. Висота символів фіксована та становить 16 пікселів. Ширина символу варіюється від 3 (1 піксель символу + 2 пікселі пробілу) до 16 пікселів. При співставленні підматриці пікселів ковзного вікна отримується символ, що має певні розміри. Тому для здійснення наступної ітерації вікно зсувається вправо на кількість пікселів, що відповідає ширині поточного символу. По завершенню рядка вікно зсувається вниз на висоту символів (16 пікселів) та вліво на початок наступного рядка.

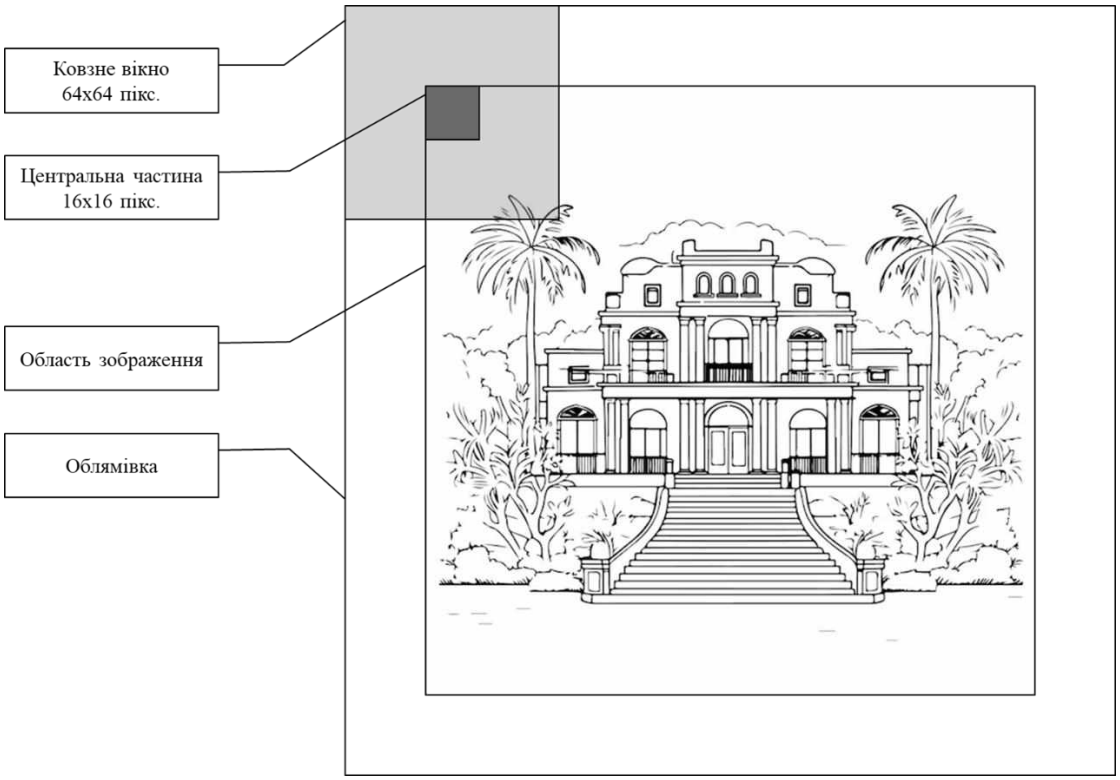


Рисунок 2.2.3 – Підхід із застосуванням ковзного вікна

Для того, щоб ковзне вікно захопило кожну ділянку вхідного зображення (коли розміри зображення не кратні розмірам вікна), до нього додається облямівка білого кольору. Використовуючи даний підхід можна отримати різні результати в залежності від початкового зсуву ковзного вікна

відносно країв зображення. Зсув має вертикальну та горизонтальну компоненти (рисунок 2.2.4). Величина кожної з них має лежати на проміжку [0, 63]. Програма використовує лише вертикальний зсув, так як вихідне символьне представлення у більшій мірі чутливе до вертикального зсуву. Причиною цього є те, що міжрядковий інтервал становить 4 пікселі, що в 4 рази більше, ніж міжсимвольний інтервал, який становить1 піксель.

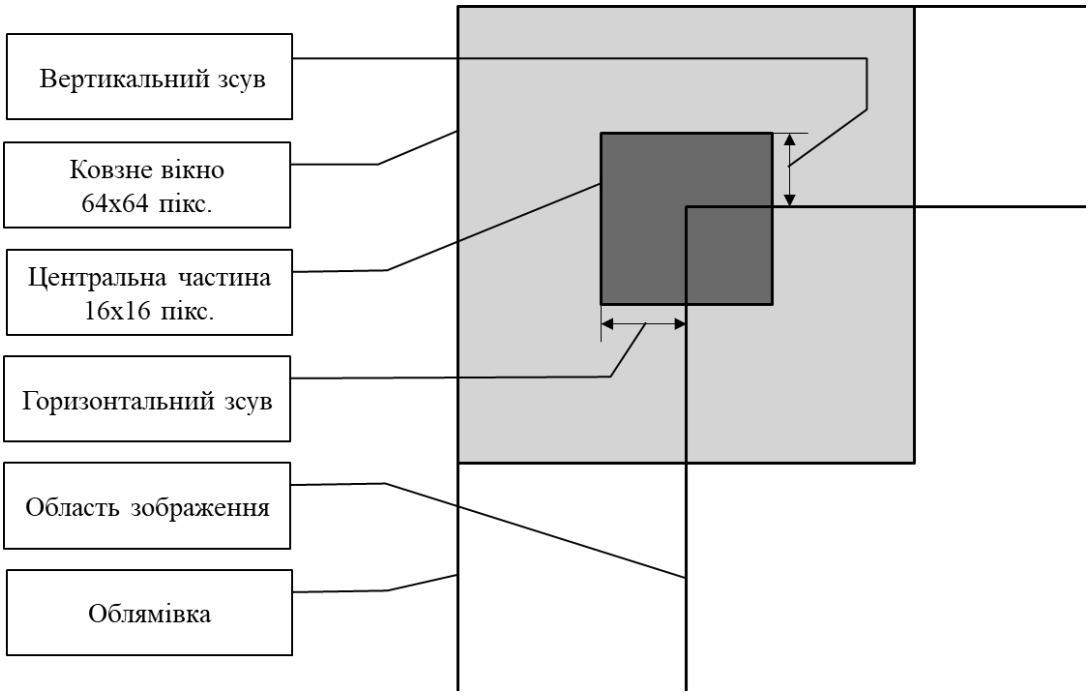


Рисунок 2.2.4 – Початковий зсув ковзного вікна

Для здійснення співставлення символу пікселям цього вікна використано згорткову нейронну мережу (CNN) [22], структурна схема якої зображена на рисунку 2.2.5.

Для тренування ШНМ використано набір даних, що складається з символів текстових документів, які містять приклади символьних малюнків ASCII Art у відповідності до зображень, отриманих рендерингом цих документів. Усі символьні малюнки створені шрифтом 12-pt MS PGothic. Тренування здійснювалося на 90% всього набора даних, 10% використовувались для валідації. Для аугментації даних використано

горизонтальний і вертикальний зсуви та гаусівський шум розмірністю в 1 піксель.

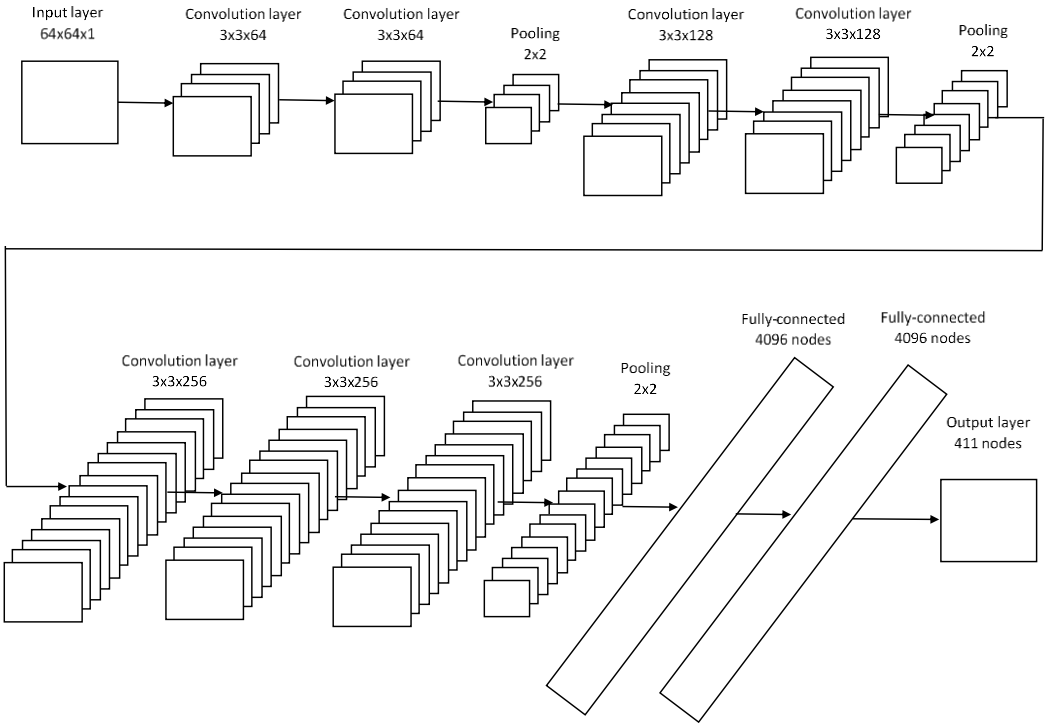


Рисунок 2.2.5 – Архітектура згорткової нейронної мережі для співставлення матриці пікселів символам

В результаті отримується текстовий файл із зображенням у символьному представленні, який при перегляді у текстовому редакторі зі встановленим вищезгаданим шрифтом буде візуально сприйматися людиною. Результат перетворення показано на рисунку 2.2.6. Більше прикладів перетворених зображень наведені в додатку А.

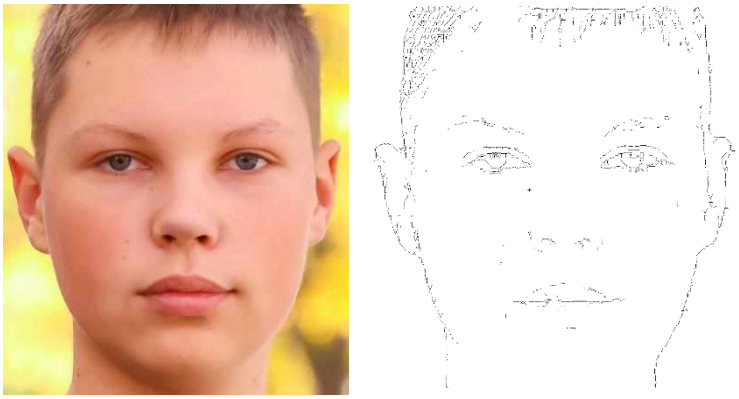


Рисунок 2.2.6 – Результат перетворення

2.3 Переваги та недоліки запропонованого методу

Переваги описаного методу, що можна виділити:

– Висока ефективність при обрізці зображення до розмірів об’єкта на ньому: при видалені фону зображення максимально обрізається таким чином, щоб повністю видалити «рамку» з фонових пікселів навколо об’єкта. Крім того етап обрізки дозволяє мінімізувати час на видалення фону та знизити обчислювальне навантаження на машину.

– Можливість як автоматичного перетворення, так і перетворення з тонким налаштуванням користувачем: користувач може пропускати деякі кроки перетворення чи виконувати їх вручну. Також метод дає можливість користувачеві підлаштовувати окремі параметри з попереднім переглядом зображення на кожному кроці.

– Гарний результат у виділенні інформативних контурів: використані алгоритми дозволяють максимально ефективно виділити інформативні контури з мінімальною залежністю від яскравості та контрастності зображення та без надлишковості даних.

– Висока ступінь стиснення: при використанні символів 16x16 пікселів коефіцієнт стиснення у порівнянні з бінарним зображенням $K \approx 3\%$. Наприклад, бінарне зображення 160x160 пікселів займає 25600 бітів або 3200 байтів, тоді як у символьному представленні воно займатиме 100 байтів.

– Результат перетворення візуально сприймається людиною при перегляді у будь-якому текстовому редакторі при встановленні необхідного шрифту.

– Можливість кросплатформеної реалізації на платформах Windows, Linux та MacOS. Крім того є можливість використання підтримуваного графічного процесора для прискорення обчислень (за його наявності у системі).

– Легкість інтеграції в СРО шляхом використання готової консольної програми.

Серед недоліків методу слід можна виділити:

– Низька швидкодія порівняно з методами без використання ШНМ.

– В процесі перетворення зображення з символьного представлення у растрове виникнуть спотворення інформації через неповну відповідність вхідних пікселів символам.

– Складності обробки зображень у символьному представленні: для виконання деяких дій, таких як поворот зображення, необхідно виконати перетворення символьного зображення у растрове.

В аспекті подальшого напрямку досліджень та вдосконалення методу та інформаційної технології перетворення растрових зображень в символьний вигляд можна зазначити наступне:

– Оптимізація запропонованого методу задля збільшення швидкості перетворення растрових зображень;

– Покращення алгоритму врахування геометричних перетворень растрових зображень, в першу чергу у випадку повороту;

– Вдосконалення методу в напрямку врахування кольорових характеристик растрових зображень.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ РОЗРОБЛЕНОГО МЕТОДУ

3.1 Консольна програма DeepAA

Було розроблено консольну програму DeepAA, що перетворює растрове зображення в символьне представлення. Взаємодія користувача з програмою здійснюється через параметри командного рядка.

Огляд параметрів програми

Для виведення списку параметрів та отримання короткої довідки з використання по кожному з них використовується параметр «--help» (або скорочений варіант «-h»). Результат роботи програми, запущеної з даним параметром зображено на рисунку 3.1.1.

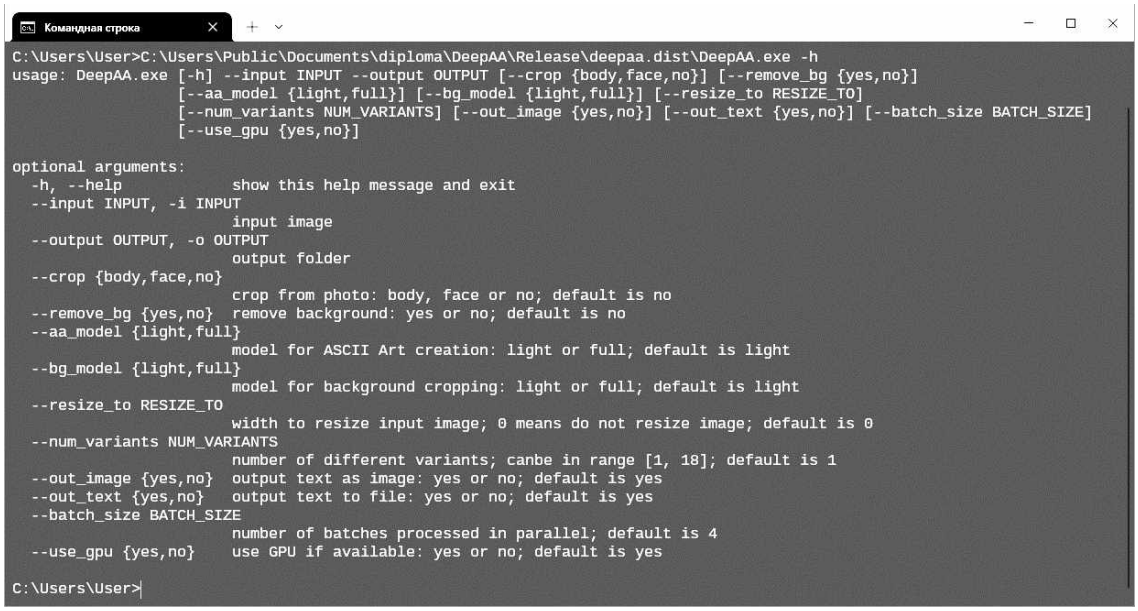


Рисунок 3.1.1 – Консольний інтерфейс розробленого програмного забезпечення

Вхідні та вихідні дані програми вказуються за допомогою параметрів «--input» («-i») та «--output» («-o»). В якості першого параметра вказується шлях до вхідного растрового зображення в одному з наведених форматів:

– Windows bitmap - *.bmp, *.dib;

– JPEG - *.jpeg, *.jpg, *.jpe;

– JPEG 2000 - *.jp2;

– Portable Network Graphics - *.png;

– WebP - *.webp;

– Portable image format - *.pbm, *.pgm, *.ppm *.pxm, *.pnm;

– Sun rasters - *.sr, *.ras;

– TIFF - *.tiff, *.tif;

– OpenEXR Image - *.exr;

– Radiance HDR - *.hdr, *.pic.

Результат роботи програма зберігає у вихідну директорію, вказану за допомогою параметру «--output». Якщо такої директорії не існує, програма здійснить спробу її створити. Помилки, що виникають під час введення та виведення обробляються шляхом завершення процесу з кодом завершення 1 та виведенням відповідного повідомлення:

– «Input must be file» – коли вхідний шлях не відповідає файлу або не існує;

– «Output must be directory» – коли вихідний шлях відповідає файлу;

– «Error accessing input/output files» – при помилці доступу до вхідних або вихідних файлів;

– «Image decode failed» – при помилці декодування зображення (якщо файл пошкоджено або його формат не підтримується).

Далі слідують параметри, що відповідають за налаштування попередньої обробки вхідного зображення:

– «--crop» – налаштовує параметри автоматичної обрізки. Приймає строку «body», «face» або «no», що відповідно позначають обрізку лише людини, лише людського обличчя та відсутність обрізки. В разі відсутності шуканого об’єкта, програма виводить попередження «No body/face detected! Full image will be used.».

– «--remove_bg» – налаштовує параметри автоматичного видалення фону. Приймає строкові значення «yes» або «no», що означають застосування чи пропуск етапу автоматичної обрізки.

– «--aa_model» – відповідає за обрання моделі нейронної мережі для перетворення зображення у символьне представлення. Приймає строкові значення «full» або «light», що означають застосування «повноцінної» чи «полегшеної» моделі. Під останньою мається на увазі аналогічна модель, що виконує ту саму задачу, має подібну архітектуру та аналогічні вхідні та вихідні шари, але відрізняється меншою кількість і (або) розмірністю прихованих шарів. Вибір «повноцінної» моделі дозволить покращити точність перетворення, однак негативно вплине на швидкодію та показники використаних ресурсів оперативного запам’ятовуючого пристрою (ОЗП) та оперативного запам’ятовуючого пристрою графічного процесора (ОЗП ГП). Використання «полегшеної» моделі позбавлене цих недоліків, але дає нижчу точність перетворення у порівнянні з попередньою моделлю.

– «--bg_model» – відповідає за обрання моделі нейронної мережі для автоматичного видалення фону. Приймає строкові значення «full» або «light», що мають аналогічну функцію та особливості застосування. В якості «повноцінної» моделі використовується оригінальна модель U^2Net, в той час як для «полегшеної» моделі – U^2Net†, описані в [23]. Структурні схеми обох моделей наведені в додатку Б.

– «--resize_to» – відповідає за розмірність растрового зображення, що приймає участь в попередній обробці. Дана опція приймає цілочисельний параметр, що може приймати значення 0 або число у проміжку [128, 4096] та позначає ширину зображення. Значення 0 позначає використання поточної розмірності зображення. Так як для перетворення на символьне представлення використовується ковзне вікно фіксованого розміру, зміна розмірів вхідного зображення буде давати різні результати перетворення з різною деталізацією. Також дана опція може бути корисною для обмеження розмірності зображення з метою запобігання надмірному споживанню ресурсів ОЗП та ОЗП ГП, а також часу, що затрачений на перетворення.

Наступна група параметрів відповідає за вихідні дані, що генерує та зберігає програма:

– «--num_variants» – відповідає за кількість варіантів перетворених вихідних зображень. Приймає ціле число на проміжку [1, 18]. Програма може генерувати різні варіанти вихідних текстових даних в залежності від початкового зсуву вікна.

– «--out_image» – відповідає за рендеринг вихідного тексту та його збереження у растровому форматі. Приймає строкові значення «yes» або «no». При першому значенні у вихідну директорію буде збережено зображення, утворене рендерингом отриманого тексту з використанням попередньо заданого шрифту та інтервалів. Назва файлу початкова з додаванням суфіксу, що позначає номер згенерованого варіанту. Вихідний формат – PNG 1bpp.

– «--out_text» – відповідає за збереження вихідної символьної послідовності у текстовий файл. Приймає строкові значення «yes» або «no». При першому значенні у вихідну директорію буде збережено файл з розширенням «.txt», що містить вихідний текст у кодуванні Shift-JIS (cp932).

Останні два параметра відповідають за організацію обчислень та швидкість здійснення перетворення:

– «--batch_size» – параметр пакетної обробки, що відповідає за кількість елементів даних, які обробляються паралельно. В контексті даної програми це означає максимальну кількість строк зображення, по яким проходять ковзні вікна одночасно. Приймає цілі значення у проміжку [1, height], де height – висота зображення. Збільшення значення даного параметра пришвидшує процес перетворення на символьне представлення за рахунок паралельної обробки. Підбір цього значення варто робити з урахуванням швидкодії ЦП або ГП, а також вільних ресурсів ОЗП та ОЗП ГП.

– «--use_gpu» – параметр, що дозволяє використовувати ГП для прискорення обчислень при автоматичній обрізці фону та перетворенні зображення на символьне представлення. Приймає строкові значення «yes» або «no». При першому значенні програма в пріоритеті використовує ГП для

обчислень при використанні ШНМ в режимі виводу. Програма дозволяє використовувати технологію Nvidia CUDA для прискорення обчислень, якщо в системі інстальовано графічний процесор, що підтримує дану технологію. Інакше, використовуються бібліотеки для прискорення обчислень на основі OpenBLAS, оптимізовані для використання центрального процесора [24]. Значення цього параметра в даному випадку буде проігноровано.

Інструментальні засоби розробки

Консольна програма створена мовою програмування Python в середовищі MS Visual Studio Code. При розробці було використано платформу Python 3.8.2 та компілятор Nuitka версії 2.2.1. Останній дозволяє траслювати код Python у C з подальшою компіляцією у виконуваний двійковий файл, що позитивно впливає на безпеку та швидкодію програмного забезпечення порівняно з використанням вбудованого інтерпретатору. Також були використані фреймворки та бібліотеки у вільному доступі:

– ONNX Runtime – платформа для запуску ШНМ в режимі виводу. Фреймворк є кросплатформеним, підтримує виконання обчислень з використанням найбільш розповсюджених апаратних та програмних засобів (OpenVINO, OpenCL, Nvidia CUDA, DirectML тощо) [25];

– OpenCV – бібліотека комп’ютерного зору, що реалізує велику кількість розповсюджених алгоритмів, які використовуються в цій області. Бібліотека є кросплатформеною та має підтримку багатьох апаратних платформ [26];

– Pillow – бібліотека для роботи з растровими зображеннями.

Реалізація програми

Розроблена консольна програма реалізує алгоритм, блок схема якого наведена в додатку В. Основна бізнес-логіка програми відображена у функції main() (рисунок 3.1.2):

– Здійснюється парсинг та обробка аргументів командного рядка за допомогою функції parse_arguments().

– Відбувається валідація шляху до вхідних та вихідних файлів. Обробляються ситуації, коли вказаний вхідний файл не існує або шлях вказує на директорію; коли вихідний шлях не відповідає директорії; при помилці доступу до вхідних або вихідних файлів; якщо вказаної вихідної директорії не існує.

– Наступним кроком є завантаження вхідного зображення та його декодування при визові функції open_image(). При помилці декодування (якщо зображення пошкоджено або його формат не підтримується) виводиться сповіщення про помилку.

– Після цього відбувається автоматична обрізка зображення за умови, коли вона ввімкнена користувачем. В залежності від об’єкта, який необхідно розпізнати, обрізка здійснюється за допомогою функцій fullbody_crop() та gace_crop(), що виконують відповідно обрізку тіла людини та людського обличчя. Якщо шуканий об’єкт не розпізнано, використовується вхідне зображення без обрізки.

– Далі здійснюється автоматичне видалення фону, якщо дана опція активована користувачем. Дану дію реалізовано у функції background_crop().

– Наступним кроком є попередня обробка зображення, яка включає в себе етапи зміни розмірів зображення, фільтрацію, бінаризацію та морфологічне стоншення. Попередня обробка виконується функцією preprocess().

– За допомогою функції convert_to_ascii() відбувається перетворення растрового зображення у символьне представлення.

– Останнім кроком є збереження результатів конвертації у вихідні файли. Виконується циклічний перебір кожного перетвореного варіанту. Для кожного з них генерується назва формату [назва вхідного файлу]_converted_[номер варіанту]. Якщо користувачем активовано опцію збереження вихідного тексту у вигляді зображення, виконується рендеринг вихідної символьної послідовності у форматі растрового зображення. Останнє

зберігаєтья у вихідний файл формату PNG. Якщо обрано опцію збереження у текстовий документ, вихідна строка записується у файл з розширенням .txt.

```
def main():
    # Parse CLI arguments
    [input_file, output_dir, crop_mode,
     remove_bg, width, num_variants,
     out_image, out_text, batch_size] = parse_arguments()

    # Check input and output files
    try:
        if not os.path.isfile(input_file):
            print(" ERROR: Input must be file")
            sys.exit(1)
        if os.path.isfile(output_dir):
            print(" ERROR: Output must be directory")
            sys.exit(1)
        elif not os.path.isdir(output_dir):
            os.makedirs(output_dir)
    except OSError:
        print(" ERROR: Error accessing input/output files")
        sys.exit(1)

    # Decode input image
    print_single_line('Preparing...')
    image = open_image(input_file)
    if image is None:
        print(" ERROR: Image decode failed")
        sys.exit(1)

    # Perform audomatic image cropping
    if crop_mode == CROP_MODE_FULLBODY:
        print_single_line('Cropping...')
        cropped = fullbody_crop(image)
        if cropped is None:
            print(" WARNING: No body detected! All image will be used.", flush=True)
        else:
            image = cropped
    elif crop_mode == CROP_MODE_FACE:
        print_single_line('Cropping...')
        cropped = face_crop(image)
        if cropped is None:
            print(" WARNING: No face detected! All image will be used.", flush=True)
        else:
            image = cropped

    # Perform automatic background substraction
    if remove_bg:
        image = background_crop(image, (255, 255, 255))

    # Image preprocessing (resizing, binarization, thinning)
    image = preprocess(image, width)

    # Conversion from raster to symbolic format
    variants = convert_to_ascii(image, num_variants, batch_size)

    # Saving results to output files
    for idx, variant in enumerate(variants):
        save_path = os.path.join(output_dir, os.path.splitext(os.path.basename(input_file))[0]+'_converted_'+str(idx+1))

        if out_image:
            render_text(variant).save(save_path+".png")

        if out_text:
            with open(save_path+".txt", 'w', encoding='cp932') as f:
                f.write(variant)

    print_single_line('Done.')
    sys.exit(0)
```

Рисунок 3.1.2 – Програмна реалізація функції main() консольної програми

Функції fullbody_crop() та gace_crop() відповідають за автоматичну обрізку тіла людини та її обличчя відповідно. Обидві функції використовують каскади Хаара для розпізнавання шуканого об’єкта та мають схожу реалізацію (рисунки 3.1.3 та 3.1.4). Функції приймають два аргументи – вхідне зображення та значення рамки навколо шуканого об’єкта, що лишається при обрізці. Бізнес-логіка, закладена у функцію, складає наступні етапи:

– Тіло кожної функції розпочинається із завантаження параметрів відповідного каскаду Хаара.

– Наступним кроком відбувається розпізнавання шуканого об’єкта та обробка ситуації, коли він не був розпізнаний. Якщо на зображенні знайдено декілька об’єктів, обрізка буде виконуватися над найбільшим із них.

– Далі здійснюється обрізка області зображення, що містить шуканий об’єкт з урахуванням границь зображення та заданої рамки.

```python
def face_crop(img, padding=50):
    # Load Haar Cascade
    cascade = cv.CascadeClassifier(os.path.join(ROOT, FACE_HAARCASCADE))

    # Perform face detection
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    faces = cascade.detectMultiScale(gray,
                                     scaleFactor=1.045,
                                     minNeighbors=6,
                                     minSize=(50, 50),
                                     flags = cv.CASCADE_FIND_BIGGEST_OBJECT | cv.CASCADE_SCALE_IMAGE)

    # Skip if no faces found
    if faces is None or len(faces) < 1:
        return None

    # Select biggest object and crop face area with padding
    x, y, w, h = faces[0]
    x1 = max(x-padding, 0)
    y1 = max(y-padding, 0)
    x2 = min(x+w+padding, img.shape[0])
    y2 = min(y+h+padding, img.shape[1])
    sub_face = img[y1:y2, x1:x2]

    return sub_face
```

Рисунок 3.1.3 – Програмна реалізація функції face_crop()

```python
def fullbody_crop(img, padding=0):
    # Load Haar Cascade
    cascade = cv.CascadeClassifier(os.path.join(ROOT, FULLBODY_HAARCASCADE))

    # Perform body detection
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    fullbody = cascade.detectMultiScale(gray,
                                        scaleFactor=1.1,
                                        minNeighbors=3,
                                        minSize=(50, 50),
                                        flags = cv.CASCADE_FIND_BIGGEST_OBJECT | cv.CASCADE_SCALE_IMAGE)

    # Skip if no bodies found
    if fullbody is None or len(fullbody) < 1:
        return

    # Select biggest object and crop body area with padding
    x, y, w, h = fullbody[0]
    x1 = max(x-padding, 0)
    y1 = max(y-padding, 0)
    x2 = min(x+w+padding, img.shape[0])
    y2 = min(y+h+padding, img.shape[1])
    sub_body = img[y1:y2, x1:x2]

    return sub_body
```

Рисунок 3.1.4 – Програмна реалізація функції fullbody_crop()

Функція background_crop() здійснює автоматичну обрізку фону зображення (рисунок 3.1.5). Для цього використовується метод на основі глибокої згорткової нейронної мережі. Функція приймає в якості аргументів вхідне зображення, колір для заливки фону та прапор обрізки області, що не містить фону. Якщо в якості кольору заливки передати значення None, функція поверне зображення з альфа-каналом у форматі BGRA. Інакше вихідне зображення матиме формат BGR. Процес видалення фону відбувається у декілька кроків:

– Видалення фону починається з ініціалізації середовища для запуску нейронної мережі в режимі виводу ONNX Runtime та завантаження моделі з файлу.

– Далі вхідне зображення обробляється у функції cv.dnn.blobFromImage() для подальшої подачі на вхідний шар ШНМ. Дана функція виконує нормалізацію зображення (приведення значень у діапазон [0, 1]), зміну розміру відповідно до розмірності вхідного шару нейронної мережі, перетворення у формат RGB та приведення до 4-вимірного представлення, перший вимір якого позначає розмірність пакету.

– Після цього відбувається прямий прохід ШНМ, внаслідок чого отримується значення на її вихідному шарі.

– Отримані значення нормалізуються та приводяться до цілочисельних у діапазоні [0, 255].

– Розмір вихідного зображення змінюється до початкового. Отримане зображення можна вважати маскою, білі області якої відповідають майбутньому вихідному зображенню, у той час як чорні області – фону, який видаляється.

– Далі з вхідного зображення виділяються кольорові канали, до яких додається альфа-канал у вигляді отриманої маски. При цьому зображення конвертується у формат RGBA.

– Наступним кроком відбувається обрізка областей, що містять фон, за умови встановлення прапора crop. Залишається прямокутна область, що містить лише об’єкт на зображенні.

– Після цього, якщо було задано колір заливки, створюється порожнє зображення, заповнене значеннями цього кольору, однакового розміру з вихідним та відбувається їх зведення.

– Останнім кроком є конвертація вихідного зображення у вхідний формат BGR.

```python
def background_crop(image, bg_color=None, crop=True):
    print_single_line('Loading background substraction model...')

    # Initialize Onnx Runtime and load background substraction model
    sess = InferenceSession(os.path.join(ROOT, bg_crop_model), providers=ort_providers)

    print_single_line('Removing background...')

    # Normalize and resize input
    blob = cv.dnn.blobFromImage(image, 1/255, (320, 320), swapRB=True, crop=False)

    # Do predict using NN
    predicted = sess.run(["output_0"], {"input_0": blob})[0][0][0]

    # Normalize output
    max_value = np.max(predicted)
    min_value = np.min(predicted)
    predicted = (predicted-min_value)/(max_value-min_value)
    predicted = (predicted*255).astype(np.uint8)

    # Create alpha channel mask
    mask = cv.resize(predicted, image.shape[1::-1], interpolation=cv.INTER_CUBIC)

    # Apply alpha channel and convert image to RGB
    b, g, r = cv.split(image)
    image = cv.merge((r, g, b, mask))

    # Crop non-transparent image area
    if crop:
        y, x = image[:,:,3].nonzero()
        minx = np.min(x)
        miny = np.min(y)
        maxx = np.max(x)
        maxy = np.max(y)
        image = image[miny:maxy, minx:maxx]

    image = Image.fromarray(image)

    # Fill background with solid color
    if not bg_color is None:
        background = Image.new('RGBA', image.size, bg_color)
        image = Image.alpha_composite(background, image)

    image = np.array(image)

    opencvImage = cv.cvtColor(image, cv.COLOR_RGB2BGR)

    return opencvImage
```

Рисунок 3.1.5 – Програмна реалізація функції background_crop()

Функція preprocess() виконує попередню обробку вхідного зображення. Її програмна реалізація наведена на рисунку 3.1.6. Функція приймає аргументи – вхідне зображення та висоту, до якої його треба привести. Обробка відбувається у чотири етапи:

– Розмір зображення змінюється таким чином, щоб його висота співпала із заданою (якщо значення відповідного аргументу не 0).

– Відбувається фільтрація зображення за допомогою двостороннього фільтру. Його параметри було підібрано експериментальним шляхом з

розрахунку на найкращий результат при використанні зображень максимальними розмірами від 256 до 840 пікселів.

– Далі зображення конвертується у відтінки сірого та здійснюється адаптивна бінаризація за Гаусом. Параметри бінаризації було підібрано аналогічним шляхом. Вихідне зображення є інвертованим (містить біле зображення об’єкта на чорному тлі).

– Останнім етапом попередньої обробки є морфологічне стоншення. Вихідне зображення містить білі лінії товщиною 1 піксель на чорному тлі. Для приведення його до вихідного представлення виконується операція інвертування.

```python
def preprocess(image, new_width=0):
    print_single_line('Preprocessing...')

    # Do resizing
    if new_width > 0:
        height, width = image.shape[:2]
        new_height = int(height * new_width / width)
        image = cv.resize(image, (new_width, new_height), cv.INTER_AREA)

    # Do filtering using bilateral filter
    image = cv.bilateralFilter(image, 5, 75, 75)

    # Do adaptive binarization
    imgray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
    thresh = cv.adaptiveThreshold(imgray, 255, cv.ADAPTIVE_THRESH_GAUSSIAN_C, cv.THRESH_BINARY_INV, 7, 5)

    # Do thinning
    skeleton = cv.ximgproc.thinning(thresh)
    skeleton = cv.bitwise_not(skeleton)

    return skeleton
```

Рисунок 3.1.6 – Програмна реалізація функції preprocess()

Функція convert_to_ascii() виконує перетворення растрового зображення на символьне представлення. Дана функція реалізує алгоритм, блок-схема якого наведена в додатку В. Програмна реалізація функції наведена на рисунку 3.1.7. В якості аргументів передається оброблене растрове зображення, кількість варіантів вихідних даних та розмір пакету. Функція повертає список строкових даних, кожна строка якого є варіантом

вихідного тексту. Перетворення на символьне представлення виконується через наступні дії:

– До вхідного зображення додається облямівка білого кольору. Вона гарантує конвертацію кожної ділянки зображення, навіть коли ковзне вікно не повністю покриває зображення (коли розміри зображення не кратні розмірам вікна).

– Вхідне зображення перетворюється у тип даних з плаваючою комою та діапазоном значень [0, 1].

– Ініціалізується платформа ONNX Runtime для запуску ШНМ та завантажується її модель. Також завантажується список символів char_list, що являє собою список об’єктів, у яких міститься інформація про символ та його ширину, а індекс у списку відповідає індексу нейрона вихідного шару ШНМ.

– Далі відбувається циклічний перебір чисел від 0 до кількості варіантів, які необхідно згенерувати. На кожній ітерації циклу відбувається зсув зображення на одну строку вниз, що забезпечує різноманіття варіантів. Обраховується кількість строк, які будуть згенеровані для поточного варіанту. Ініціалізується порожній рядок text, який буде заповнюватися символами в процесі конвертації зображення у текст.

– Внутрішній цикл, що перебирає номери строк з інтервалом, рівним розміру пакету, слугує для передачі наступних строк для пакетної обробки. Створюються масиви lines, w і penalty, які використовуються для зберігання поточних строк тексту, ширини тексту, що обробляється, та змінних штрафів для запобігання непотрібного повторення пробілів.

– Всередині циклу відбувається пакетна обробка строк. Запускається нескінченний цикл while True, в якому створюються нові пакети вхідних даних для ШНМ, що забезпечує постійну обробку окремих строк у поточному пакеті.

– Вхідні дані для нейронної мережі збираються у поточному пакеті. Для кожної лінії створюється вікно зображення (window), яке додається до списку inputs.

– Здійснюється перевірка, чи є ще вхідні дані для обробки. Якщо даних більше немає, до результуючого тексту додаються непорожні строки, і цикл завершується.

– Вхідні дані об'єднуються у єдиний масив inputs для подачі у нейронну мережу, що дозволяє виконати прогнозування для всіх вікон одночасно. Виконується прогнозування за допомогою ШНМ, що повертає прогнозовані оцінки символів outputs для кожного вікна.

– Результати прогнозування обробляються. Знижується ймовірність повторення пробілів за допомогою масиву прапорів penalty, обираються символи з найвищим рейтингом, і додаються до поточних строк тексту.

– Додані строки тексту об'єднуються у кінцевий текстовий варіант (text) і додаються до вихідного списку варіантів тексту text_variants.

```python
def convert_to_ascii(image, variants, batch_size=1):
    # Add margin to input image
    height, width = image.shape[:2]
    image_new = np.ones([height+2*MARGIN+GLYPH_SIZE, width+2*MARGIN+GLYPH_SIZE], dtype=np.uint8)*255
    image_new[MARGIN:MARGIN+height, MARGIN:MARGIN+width] = image
    height, width = image_new.shape[:2]

    # Convert to float array with values in range [0, 1]
    image_new = (image_new.astype(np.float32)) / 255

    # Load NN model and initialize Onnx Runtime session
    print_single_line('Loading ASCII Art creation model...')
    sess = InferenceSession(os.path.join(ROOT, transformer_model), providers=ort_providers)

    # Load "prediction to char and it's width" lookup table
    with open(os.path.join(ROOT, CHAR_TABLE)) as f:
        char_list = json.loads(f.read())

    text_variants = []

    for slide in range(variants):
        # Shift image down by 1 line
        new_line = np.ones([1, width])
        image_new = np.concatenate([new_line, image_new], axis=0)

        num_lines = (height - WINDOW_SIZE) // GLYPH_SIZE
        batch_size = min(batch_size, num_lines)

        text = ""

        # Iterate over each line of input image
        for h_start in range(0, num_lines+batch_size, batch_size):
            progress = (slide * num_lines + h_start)*100//(variants * num_lines)
            print_single_line(f"\rConverting: {progress} %")

            # Start processing next {batch_size} lines
            lines = [""]*batch_size
            w = [0]*batch_size
            penalty = [True]*batch_size
            while True:
                # Create new batch of input data
                inputs = []
                indices = []
                for i in range(batch_size):
                    # Current line number
                    h = h_start+i
                    if w[i] <= width-WINDOW_SIZE and h < num_lines:
                        # Crop sliding window area from input image
                        window = image_new[h*GLYPH_SIZE:h*GLYPH_SIZE+WINDOW_SIZE, w[i]:w[i]+WINDOW_SIZE]
                        window = window.reshape([1, WINDOW_SIZE, WINDOW_SIZE, 1])
                        # Save inputs and their indices in batch
                        inputs.append(window)
                        indices.append(i)

                # Quit on end of image lines
                if len(inputs) == 0:
                    for i in range(batch_size):
                        # Save non-empty lines
                        if len(lines[i]) > 0:
                            text += lines[i]+"\n"
                    break

                # Stack several inputs to create a batch
                inputs = np.vstack(inputs, dtype=np.float32)

                # Perform prediction using CNN
                outputs = sess.run(["predictions"], {"input_1": inputs})[0]

                for out, idx in zip(outputs, indices):
                    # Reduce space symbol (index 1) score if previous symbol was already space
                    if penalty[idx]: out[1] = 0
                    # Find index of char with highest score
                    predict = np.argmax(out)
                    # Remember if last symbol is space
                    penalty[idx] = (predict == 1)

                    # Get char and it's width from lookup table
                    char = char_list[predict]
                    lines[idx] += char["char"]
                    w[idx] += char["width"]

        text_variants.append(text)

    print_single_line('Converting: 100 %')

    return text_variants
```

Рисунок 3.1.7 – Програмна реалізація функції convert_to_ascii()

3.2 Графічна оболонка DeepAA-GUI

Також була створена графічна оболонка DeepAA-GUI для консольної програми DeepAA. Вона являє собою програму з графічним інтерфейсом, що використовує графічні елементи керування для полегшення процесу перетворення. Вона не додає нового функціоналу стосовно процесу перетворення зображень, тому є додатковим компонентом для консольної програми. Вигляд графічного інтерфейсу програми наведено на рисунку 3.2.1.

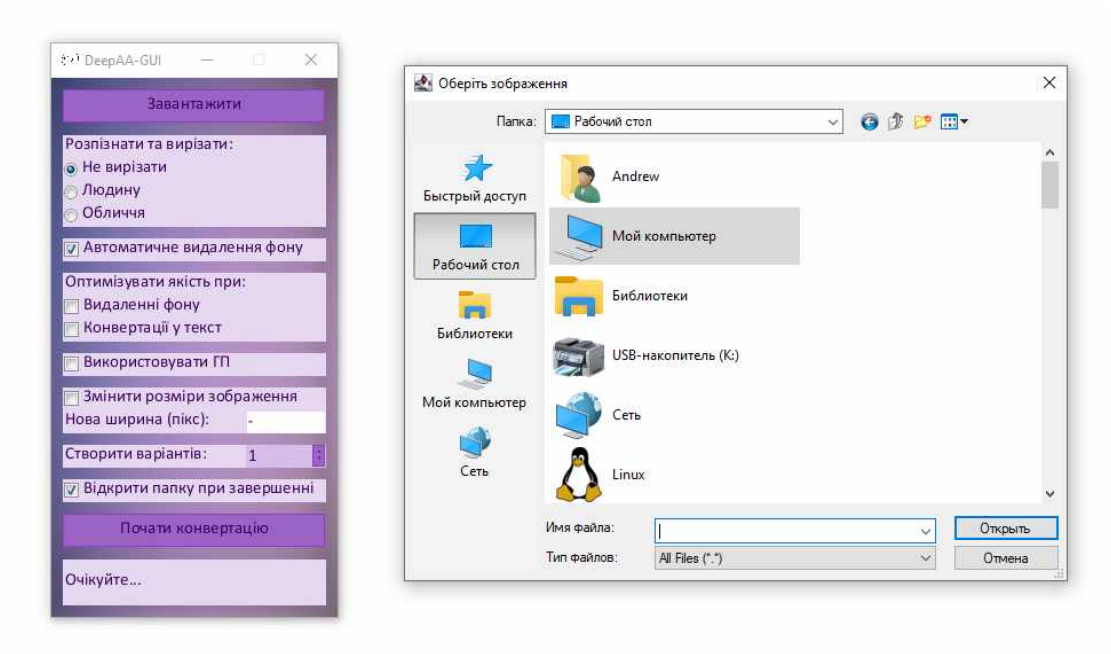


Рисунок 3.2.1 – Графічний інтерфейс розробленого програмного забезпечення

Огляд графічного інтерфейсу

Графічний інтерфейс програми складається з одного вікна, на якому розміщено елементи GUI (рисунок 3.2.2):

– Button – кнопки «Завантажити» та «Почати конвертацію»;

– Label – написи «Розпізнати та вирізати», «Оптимізувати якість при», область виводу статусу конвертації;

– Input – поле введення «Нова ширина (пікс)»;

– Checkbox – поля «Автоматичне введення фону», «Видаленні фону», «Конвертації у текст», «Використовувати ГП», «Відкрити папку при завершенні»;

– Radio – елементи «Не вирізати», «Людину», «Обличчя»;

– Combo – випадаючий список «Створити варіантів».

Мова інтерфейсу українська. Кольорова тема незмінна. Узгодженість елементів графічного інтерфейсу проявляється у зміні активності елементів керування при відсутніх вхідних даних та під час здійснення конвертації. Завдяки кросплатформеній реалізації движка GUI, вигляд елементів графічного інтерфейсу є однаковим при виконанні застосунку в різних середовищах та на різних операційних системах. Поле «Відкрити папку при завершенні» відповідає за показ вихідної директорії у провіднику Windows після вдалої конвертації. У інших операційних системах дане поле неактивне та відповідна функція недоступна. Вибір вхідних та вихідних файлів здійснюється за допомогою стандартних діалогових вікон графічної оболонки операційної системи.

Рисунок 3.2.2 – Елементи графічного інтерфейсу програми DeepAA-GUI

Інструментальні засоби розробки

Програма створена мовою Java у середовищі Processing IDE. Під час розробки використано програмне забезпечення у вільному доступі:

– платформу OpenJDK версії 17.0.8

– бібліотеку G4P та однойменний WYSIWYG-конструктор графічного інтерфейсу.

Реалізація програми

У програмному коді графічної оболонки здійснюється обробка подій взаємодії користувача з елементами інтерфейсу та запис відповідних значень у поля основного класу. При натисканні кнопки «Почати конвертацію» виконується функція execute_deepaa(), що запускає консольну програму DeepAA з заданими параметрами та повертає значення коду завершення процесу. Також здійснюється зчитування стандартних потоків виводу та помилок процесу та оновлення статусу у нижній області вікна. Програмний код цієї функції наведено на рисунку 3.2.3.

```
int execute_deepaa() {
  String file[] = { dataPath("DeepAA/DeepAA"),
                    "--input", "\"" + input + "\"",
                    "--output", "\"" + output + "\"",
                    "--crop", crop,
                    "--remove_bg", remove_bg,
                    "--num_variants", variants,
                    "--aa_model", aa_model,
                    "--bg_model", bg_model,
                    "--resize_to", resize,
                    "--out_text", out_text,
                    "--out_image", out_image,
                    "--batch_size", batch_size,
                    "--use_gpu", use_gpu};

  println("Executing " + join(file, ' '));

  int exit_code = -1;

  try {
    Process p = Runtime.getRuntime().exec(file, null, new File(dataPath("DeepAA")));
    InputStreamReader stdout_is = new InputStreamReader(p.getInputStream());
    InputStreamReader stderr_is = new InputStreamReader(p.getErrorStream());
    BufferedReader stdout_reader = new BufferedReader(stdout_is);
    BufferedReader stderr_reader = new BufferedReader(stderr_is);
    String line = "";

    println("===| PROCESS STARTED |===");

    while((line = stdout_reader.readLine()) != null)
    {
      if(!line.trim().isEmpty())
      {
        println(line);
        for(int i = 0; i < dict.length; i++)
          line = line.replace(dict[i][0], dict[i][1]);
        label7.setText(line);
      }
    }

    exit_code = p.waitFor();

    while((line = stderr_reader.readLine()) != null)
    {
      println(line);
    }

    println("===| PROCESS STOPPED |===");
    println("Exit code: "+exit_code);
  } catch(IOException exception) {
    exception.printStackTrace();
  } catch(InterruptedException exception) {
    exception.printStackTrace();
  }

  return exit_code;
}
```

Рисунок 3.2.3 – Програмна реалізація функції execute_deepaa()

3.3 Інсталяційний пакет

Дві розроблені програми збираються в єдиний інсталяційний пакет, що установлює консольну програму за замовчуванням та надає можливість користувачу обрати установку графічної оболонки в якості додаткового

компонента. Також є опція додавання шляху до директорії з виконуваним файлом консольної програми до змінної PATH для швидкого доступу до цієї програми. Для графічної оболонки програма-інсталятор створює ярлики для швидкого доступу на робочому столі та меню Пуск. На операційній системі Windows інсталяція лише консольної програми займає 1126 МБ об’єму на носії. Встановлення графічної оболонки вимагає додаткових 258 МБ. Розмір інсталяційного пакету становить 654 МБ. Зовнішній вигляд основних етапів інсталяції розробленого програмного забезпечення наведено на рисунку 3.3.1. Для створення інсталяційного пакету використано програмне забезпечення Advanced Installer від Caphyon LTD, яке дозволяє створювати програми-інсталятори з просунутими можливостями налаштування їх бізнес-логіки.

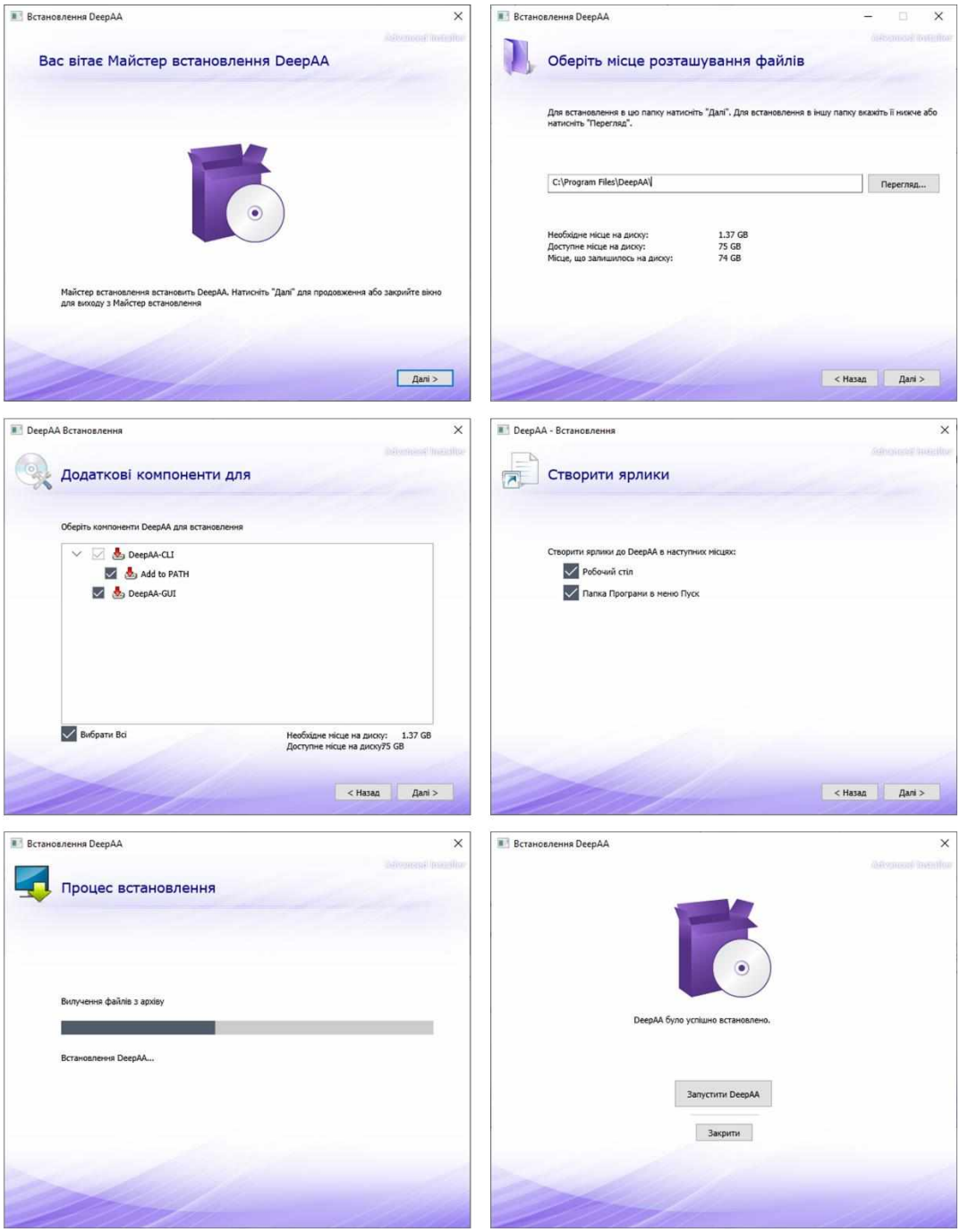


Рисунок 3.3.1 – Зовнішній вигляд інтерфейсу інсталяційного пакета для розробленого програмного забезпечення

ВИСНОВКИ

В ході проведеного дослідження було розроблено ефективне та гнучке програмне забезпечення для перетворення растрового зображення у символьне представлення. Таким чином, основна мета дипломної роботи – досягнута. Всі поставлені задачі були вирішені. В результаті проведеного дослідження було отримано наступні основні результати:

1. Здійснено огляд основних методів обробки та перетворення растрових зображень, в результаті якого визначено їх основні особливості. Сформульовано основні вимоги до альтернативного методу структурної трансформації.

2. Запропоновано метод перетворення растрового зображення у символьне представлення. Він включає етапи обрізки та видалення фону, фільтрації, бінаризації, стоншення та трансформації у символьне представлення. Останнє здійснюється за допомогою згорткової нейронної мережі, що дозволяє створити символьне представлення для кожної частини зображення. За результатами дослідження визначено основні переваги та недоліки запропонованого методу та символьного представлення взагалі.

3. На основі запропонованого методу створено програмне забезпечення для перетворення растрового зображення у символьне представлення. Воно виявляється ефективним та зручним для використання, а його можливості розширюються за рахунок наявності параметрів для ручного налаштування, що дозволяє виконувати широкий спектр завдань з перетворення растрових зображень у символьні рисунки, таких як публікація малюнків в соцмережах та використання в системах розпізнавання образів.

4. Визначені напрямки подальших досліджень та вдосконалення методів та інформаційної технології перетворення растрових зображень в символьне представлення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ASCII-графіка – Wikipedia. URL: https://uk.wikipedia.org/wiki/ASCII-графіка (дата звернення: 01.06.2024).
2. Захожай О. І., Крохмаль А. В. Програмне забезпечення для перетворення растрового зображення на символьний рисунок за структурним принципом - «ТАК»: телекомунікації, автоматика, комп'ютерно-інтегровані технології: зб. доповідей Всеукр. наук.-практ. конф. молодих вчених, 25-26 листопада 2020 р. / ДВНЗ «ДонНТУ; відп. ред. Г.В. Ступак. – Покровськ: ДВНЗ «Дон- НТУ», 2020. – 167 с.
3. Крохмаль А. В., Захожай О. І. Програмне забезпечення для перетворення растрового зображення у символьне представлення – ІТ-Ідея – 2023: збірник науково-практичних праць. – Київ: вид-во Східноукр. ун-ту ім. В. Даля, 2023. – 130 с.
4. Крохмаль А. В., Захожай О. І. Селекція інформативних даних шляхом перетворення растрового зображення в символьне представлення для інформаційних систем розпізнавання образів - Майбутній науковець – 2020 : матеріали всеукр. наук.-практ. конф. з міжнар. участю 4 груд. 2020 р., м. Сєвєродонецьк. / укладач В. Ю. Тарасов – Сєвєродонецьк : Східноукр. нац. ун-т ім. В. Даля, 2020. – 338 с.
5. Sun Y, Wang X, Tang X. Deep Convolutional Network Cascade for Facial Point Detection. Computer Vision and Pattern Recognition (CVPR). NY. : IEEE Conference on. IEEE. 2013. – pp. 3476-3483.
6. Bataineh B., Omar, K. An adaptive local binarization method for document images based on a novel thresholding method and dynamic windows. Pattern Recognit. Lett. NY.: IEEE Conference on. IEEE. 2011. vol. 32. – pp. 1805-1813.
7. Захожай О. І., Крохмаль А. В. Екстенсіональний підхід до розпізнавання растрових зображень на основі їх символьних перетворень - Наукові

вісті Далівського університету. №25. 2023р. DOI: 10.33216/2222-3428-2023-25-2.

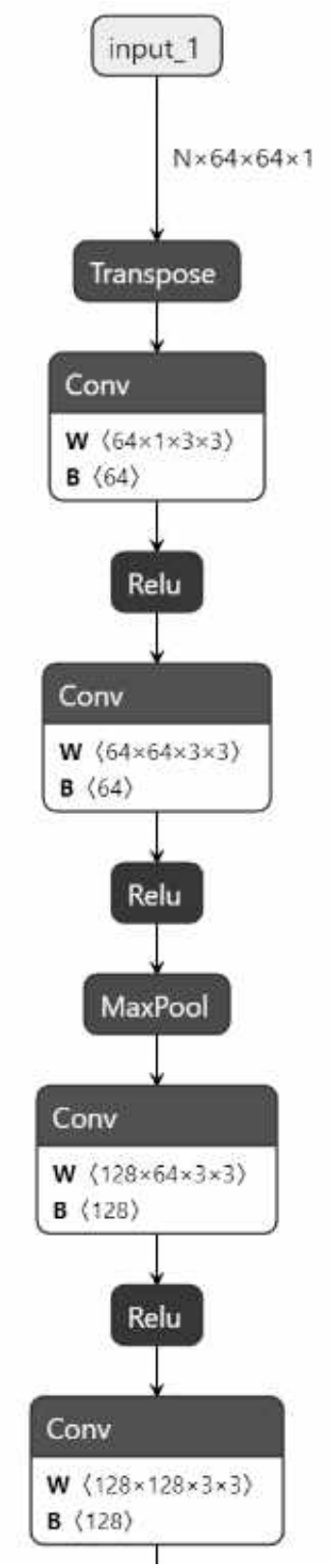
8. Fujimoto T. R., Kawasaki T., Kitamura K. Canny-Edge-Detection / Rankine-Hugoniot-conditions unified shock sensor for inviscid and viscous flows. Journal of Computational Physics. 2019. vol. 396 – pp. 264-279.
9. Nixon M. S., Aguado A. S. Basic image processing operations. Feature Extraction & Image Processing for Computer Vision. December 2012 – pp. 83-136.
10. Image filtering and morphology, E.R. Davies // Computer Vision (Fifth Edition) Principles, Algorithms, Applications, Learning. 2018 – pp. 78-84. DOI: 10.1016/B978-0-12-809284-2.00003-4.
11. Hafiz A. M., Bhat G. M. A Survey on Instance Segmentation: State of the art. International Journal of Multimedia Information Retrieval. K.: University of Kashmir. 2020. vol. 9 – pp. 171-189.
12. Takeuchi Y., Takafuji D., Ito Y,, Nakano K. ASCII Art Generation using the Local Exhaustive Search on the GPU. Conference: Proceedings of the 2013 First International Symposium on Computing and Networking. H.: Hiroshima University. 2013. DOI:10.1109/CANDAR.2013.35.
13. Structure-based ASCII Art - Xuemiao Xu, Linling Zhang, Tien-Tsin Wong. The Chinese University of Hong Kong. ACM Transactions on Graphics. 2010. 07. Vol. 29, Iss. 4. DOI: 10.1145/1833351.1778789.
14. ASCII Art Maker – Softpedia. URL: https://www.softpedia.com/get/Multimedia/Graphic/Image-Convertors/ASCII-Art-Maker.shtml (дата звернення: 01.06.2024).
15. Characterizer – Softpedia. URL: https://www.softpedia.com/get/Multimedia/Graphic/Image-Convertors/Characterizer.shtml (дата звернення: 01.06.2024).
16. ImageToText – Softpedia. URL: https://www.softpedia.com/get/Multimedia/Graphic/Graphic-Others/ImageToText.shtml (дата звернення: 01.06.2024).

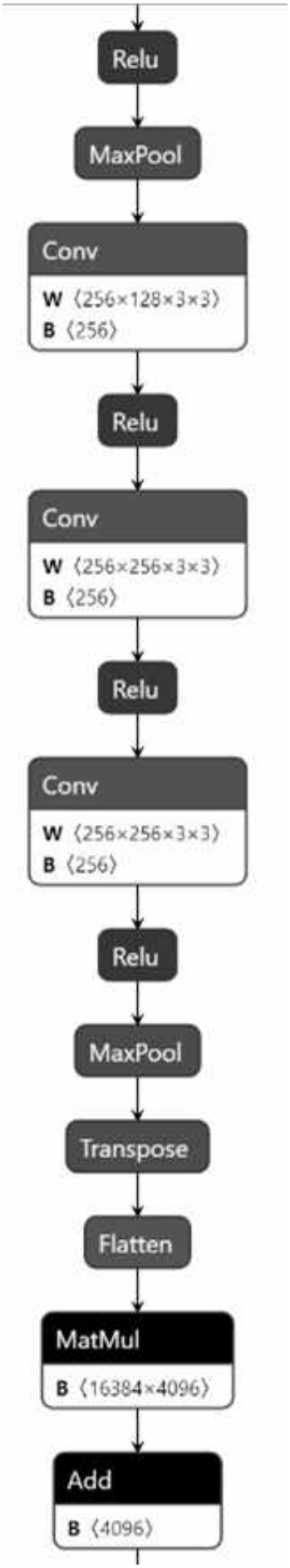
17. ASCII Generator 2. URL: https://ascgendotnet.jmsoftware.co.uk (дата звернення: 01.06.2024).
18. Asciiart.club – URL: https://asciiart.club (дата звернення: 01.06.2024).
19. DeepAAonWeb. URL: https://tar-bin.github.io/DeepAAonWeb (дата звернення: 01.06.2024).
20. Face Recognition using Haar Cascade Classifier. International Journal for Modern Trends in Science and Technology, 7(01). January 2021. 85-87. DOI: 10.46501/IJMTST070119.
21. Chen, L., Papandreou, G., Kokkinos, I., Murphy, K., Yuille, A. L. DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. 2017.
22. Akiyama, O. ASCII Art Synthesis with Convolutional Networks. Faculty of Medicine, Osaka University. Materials of the Conference: Proceedings of the 2013 First International Symposium on Computing and Networking, Long Beach, California, USA. 4-9 December 2017.
23. U2-Net: Going Deeper with Nested U-Structure for Salient Object Detection, Xuebin Qin, Zichen Zhang, Chenyang Huang, Masood Dehghan, Osmar R. Zaiane, Martin Jagersand, DOI: 10.48550/arXiv.2005.09007
24. Wu, Y., Liu, L., Pu, C., Sahin, S., Wei, W., & Zhang, Q. (2019). A Comparative Measurement Study of Deep Learning as a Service Framework. IEEE Transactions on Services Computing. 18 July 2019. 1-15. DOI: 10.1109/TSC.2019.2928551.
25. ONNX Runtime. URL: https://onnxruntime.ai (дата звернення: 01.06.2024).
26. OpenCV - Open Computer Vision Library. URL: https://opencv.org (дата звернення: 01.06.2024).

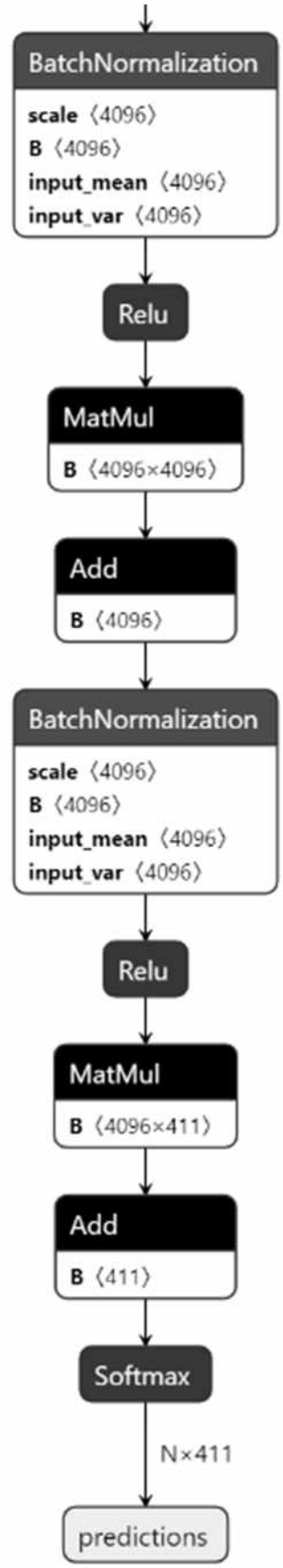
ДОДАТКИ

Додаток А. Приклади перетворених зображень

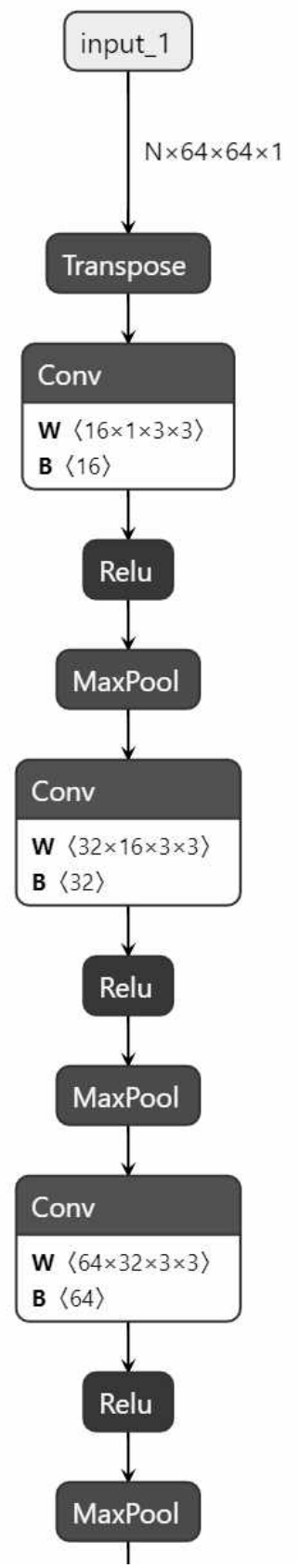
Додаток Б. Структурна схема «повноцінної» моделі нейронної мережі для перетворення зображення у символьне представлення

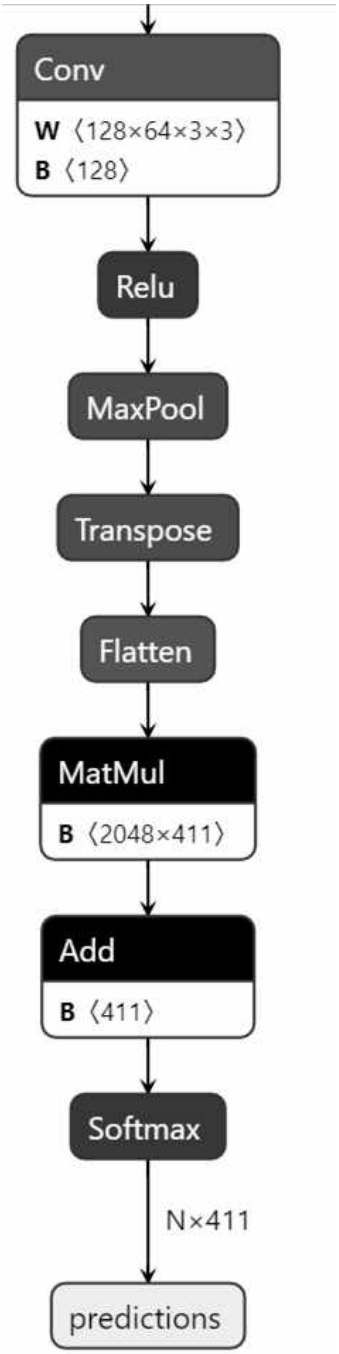



Relu
MaxPool
Conv
W 〈256×128×3×3〉
B 〈256〉
Relu
Conv
W 〈256×256×3×3〉
B 〈256〉
Relu
Conv
W 〈256×256×3×3〉
B 〈256〉
Relu
MaxPool
Transpose
Flatten
MatMul
B 〈16384×4096〉
Add
B 〈4096〉

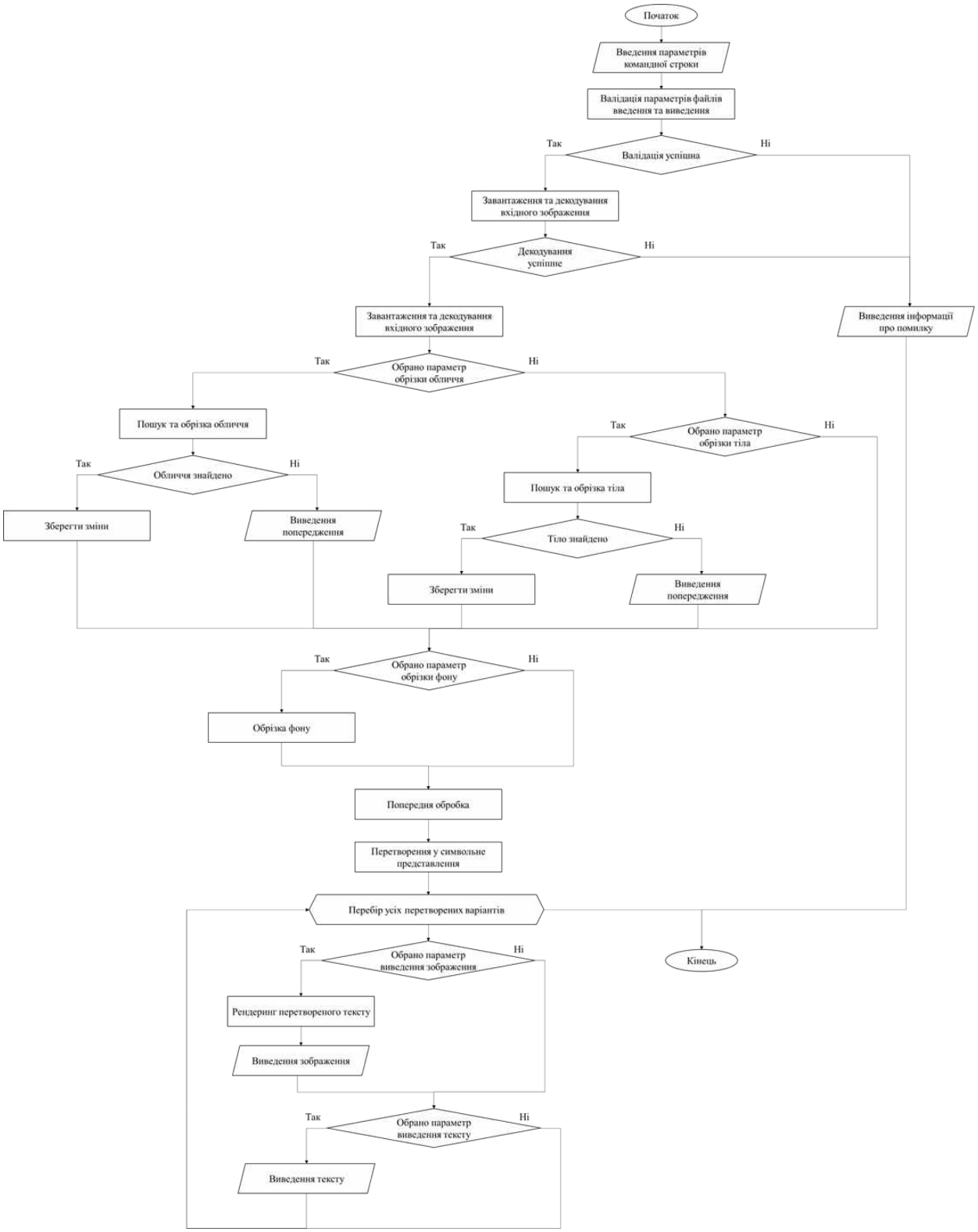

BatchNormalization
scale ⟨4096⟩
B ⟨4096⟩
input_mean ⟨4096⟩
input_var ⟨4096⟩
Relu
MatMul
B ⟨4096×4096⟩
Add
B ⟨4096⟩
BatchNormalization
scale ⟨4096⟩
B ⟨4096⟩
input_mean ⟨4096⟩
input_var ⟨4096⟩
Relu
MatMul
B ⟨4096×411⟩
Add
B ⟨411⟩
Softmax
N×411
predictions

Структурна схема «полегшеної» моделі нейронної мережі для перетворення зображення у символьне представлення




Conv
W 〈128×64×3×3〉
B 〈128〉
Relu
MaxPool
Transpose
Flatten
MatMul
B 〈2048×411〉
Add
B 〈411〉
Softmax
N×411
predictions

Додаток В. Блок-схема алгоритму розробленого програмного забезпечення



Блок-схема алгоритму перетворення растрового зображення у символьне представлення