НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2024 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Бібліотека для визначення контурів рельєфу з супутникових фотографій

Виконав студент IV курсу, групи ІТ-01
(шифр групи)

Філянін Нікіта Сергійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник ст. викл., PhD, Стельмах О. П. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент к.т.н., доцент, Сперкач М. О. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2024

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2024 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Філяніну Нікіті Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Бібліотека для визначення контурів рельєфу з супутникових фотографій

керівник проєкту Стельмах Олександр Петрович, ст. викл., PhD
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «27»квітня 2024 р. №2112-с

2. Термін подання студентом проєкту « 17 » червня 2024 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Передпроєктне обстеження предметної області: аналіз предметної області, аналіз існуючих рішень, опис бізнес-процесів, постановка задачі.
2) Розроблення вимог до програмного забезпечення: варіанти використання програмного забезпечення, розроблення функціональних та нефункціональних вимог.
3) Конструювання та розроблення програмного забезпечення: архітектура програмного забезпечення, обґрунтування вибору засобів розробки, конструювання програмного забезпечення, аналіз безпеки даних.
4) Аналіз якості та тестування: аналіз якості ПЗ, опис процесів тестування, опис контрольного прикладу.
5) Розгортання та супровід програмного забезпечення: розгортання програмного забезпечення, супровід програмного забезпечення.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань _____

2) Схема структурна діяльності_____

3) Діаграма пакетів бібліотеки_____

4) Діаграма структури сховища_____

6. Консультанти розділів проєкту

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

7. Дата видачі завдання «11» березня 2024 року _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	11.03.2024	
2	Аналіз існуючих методів розв’язання задачі	18.03.2024	
3	Постановка та формалізація задачі	25.03.2024	
4	Розробка інформаційного забезпечення	07.04.2024	
5	Алгоритмізація задачі	15.04.2024	
6	Обґрунтування вибору використаних технічних засобів	23.04.2024	
7	Розробка програмного забезпечення	05.05.2024	
8	Налагодження програми	14.05.2024	
9	Виконання графічних документів	21.05.2024	
10	Оформлення пояснювальної записки	02.06.2024	
11	Подання ДП на попередній захист	02.06.2024	
12	Подання ДП рецензенту	10.06.2024	
13	Подання ДП на основний захист	14.06.2024	

Студент _____ Нікіта ФІЛЯНІН

(підпис) (ініціали, прізвище)

Керівник _____ Олександр СТЕЛЬМАХ

(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з п’яти розділів, містить 34 таблиці, 28 рисунків та 24 джерела – загалом 65 сторінок.

Дипломний проєкт присвячений розробці бібліотеки для визначення контурів рельєфу з супутникових фотографій.

Метою розробки є підвищення зручності виконання процесу визначення контурів рельєфу з супутникових фотографій, за рахунок об’єднання методів обробки та інструментів для автоматизованого аналізу зображень в одну Python бібліотеку.

Об'єкт дослідження: розробка бібліотеки для визначення контурів рельєфу з супутникових фотографій.

Предмет дослідження: бібліотека для визначення контурів рельєфу з супутникових фотографій, інструменти та методи для автоматизації процесу обробки зображень.

У першому розділі було проведено аналіз предметної області та існуючих рішень, описано бізнес-процеси та сформульовано задачу.

Другий розділ присвячений розробленню вимог до програмного забезпечення, визначенню варіантів використання, формулюванню функціональних та нефункціональних вимог.

У третьому розділі було розглянуто підходи до конструювання та розроблення програмного забезпечення, описано архітектуру, обґрунтовано вибір засобів розробки, проведено аналіз безпеки даних.

Четвертий розділ присвячений аналізу якості розробленого програмного забезпечення, опису процесів його тестування та контрольного прикладу.

У п’ятому розділі було здійснено опис способу розгортання програмного забезпечення та його супроводу.

КЛЮЧОВІ СЛОВА: БІБЛІОТЕКА, PYTHON, ВИЗНАЧЕННЯ КОНТУРІВ, РЕЛЬЄФ, СУПУТНИК, ФОТОГРАФІЇ, ОБРОБКА ЗОБРАЖЕНЬ.

ABSTRACT

The explanatory note of the diploma project consists of five chapters and contains 34 tables, 28 figures, and 24 sources - a total of 65 pages.

The diploma project is dedicated to developing a library for determining relief contours from satellite photos.

The purpose of the development is to increase the convenience of performing the process of determining relief contours from satellite photos by combining processing methods and tools for automated image analysis into one Python library.

The object of research: development of a library for determining relief contours from satellite images.

The subject of research: a library for determining relief contours from satellite images, tools, and methods for automating image processing.

The first section analyzes the subject area and existing solutions, describes business processes, and formulates the problem.

The second section is devoted to developing software requirements, identifying use cases, and formulating functional and non-functional requirements.

The third section describes approaches to software design and development, describes the architecture, justifies the choice of development tools, and analyzes data security.

The fourth section analyzes the quality of the developed software and describes its testing processes and a test case.

The fifth section describes the method of software deployment and its maintenance.

KEYWORDS: LIBRARY, PYTHON, CONTOUR DETECTION, RELIEF, SATELLITE, PHOTOS, IMAGE PROCESSING.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІТ-0321.045490.01.91	Технічне завдання	12	
3	А4	КПІ.ІТ-0321.045490.02.81	Пояснювальна записка	65	
4	А4	КПІ.ІТ-0321.045490.03.12	Текст програми	12	
5	А4	КПІ.ІТ-0321.045490.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІТ-0321.045490.05.34	Керівництво користувача	12	
7	А3	КПІ.ІТ-0321.045490.06.99	Графічний матеріал	4	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІТ-0321.045490.00.90		
Розроб.		Філянін Н.С.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Стельмах О.П.					
Н.контр.		Вітковська І.І.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІТ-01	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2024 р.

БІБЛІОТЕКА ДЛЯ ВИЗНАЧЕННЯ КОНТУРІВ РЕЛЬЄФУ З СУПУТНИКОВИХ ФОТОГРАФІЙ

Технічне завдання

КПІ.ІТ-0321.045490.01.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Олександр СТЕЛЬМАХ

Нормоконтроль:

_____ Ірина ВІТКОВСЬКА

Виконавець:

_____ Нікіта ФІЛЯНІН

Київ – 2024

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Бібліотека для визначення контурів рельєфу з супутникових фотографій.

Галузь застосування:

Наведене технічне завдання поширюється на розробку програмного забезпечення «Бібліотека для визначення контурів рельєфу з супутникових фотографій» КПІ.ІТ-0321.045490, котре використовується для виявлення контурів геологічних структур (семантичних класів) з растрових зображень та призначена для використання геодезистами і планетарних вченими, які досліджують клімат і географію Марса.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки бібліотеки для визначення контурів рельєфу з супутникових фотографій є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для організації та автоматизації процесу обробки растрових супутникових зображень з метою визначення контурів рельєфу геологічних структур.

Метою розробки є підвищення зручності виконання процесу визначення контурів рельєфу з супутникових фотографій, за рахунок об’єднання методів обробки та інструментів для автоматизованого аналізу зображень в одну Python бібліотеку.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функцій:

4.1.1 Для користувача:

– введення параметрів за допомогою інтерфейсу командного рядка;
– вибір зображення для обробки;
– завантаження зображення з серверу PDS (Planetary Data System);
– вибір директорії для сховища даних;
– вибір конфігураційного файлу;
– вибір моделі нейронної мережі UNet;
– встановлення параметрів для стискання і конвертації зображення;
– перегляд метаданих обраного зображення;
– встановлення параметрів для моделі UNet;
– встановлення параметрів для очистки бінарної маски;
– встановлення параметрів для отримання контурів;
– перегляд результатів обробки у вигляді списку координат точок, які формують знайдений контур;
– перегляд результатів обробки у вигляді зображення із графічно нанесеним контуром.

4.1.2 Додаткові вимоги:

– список координат точок, які формують контур має зберігатись у форматі NPY;
– користувач має змогу редагувати конфігураційний файл, вказуючи свої значення параметрів;
– стиснуті версії оригінальних зображень мають зберігатись в окремій директорії.

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача. Забезпечити цілісність даних при локальному збереженні.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються.

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

Мінімальна конфігурація технічних засобів:

– тип процесору: Intel Core i5;
– об‘єм ОЗП: 8 Гб;
– наявність 30 Гб вільного місця на фізичному накопичувачі;
– підключення до мережі Інтернет зі швидкістю від 20 мегабіт.

Рекомендована конфігурація технічних засобів:

– тип процесору: AMD Ryzen 5 або Intel Core i7;
– об‘єм ОЗП: 16 Гб;
– наявність 50 Гб вільного місця на фізичному накопичувачі;
– підключення до мережі Інтернет зі швидкістю від 100 мегабіт;
– наявність графічного процесору NVIDIA із підтримкою CUDA.

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем Windows та Linux.

4.5.1 Вимоги до вхідних даних

Вхідні дані повинні бути представлені в наступному форматі: растрові зображення формату JP2.

4.5.2 Вимоги до вихідних даних

Результати повинні бути представлені в наступному форматі: списки координат точок, які формують визначені контури, зберігаються в файлах формату NPY. Зображення із графічно визначеними контурами зберігаються в форматі PNG.

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування Python.

4.5.4 Вимоги до середовища розробки

Розробку виконати за допомогою середовища розробки Visual Studio Code, із використанням Conda в якості інструменту для управління віртуальними середовищами Python.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді завантаженого готового проєкту в репозиторій на платформі GitHub. Код має також бути наведено в документі «Текст програми».

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Для використання програмного забезпечення передбачається наявність встановленого інструменту Conda.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

– пояснювальна записка;
– технічне завдання;
– текст програми;
– програма та методика тестування;
– керівництво користувача;

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

– схема структурна варіантів використання;
– схема структурна діяльності;
– діаграма пакетів бібліотеки;
– діаграма структури сховища.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	11.03	
2.	Розробка технічного завдання	21.03	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	26.03	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	21.04	Схема структурна програмного забезпечення та специфікація компонентів
5.	Програмна реалізація програмного забезпечення	03.05	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	12.05	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	16.05	Пояснювальна записка
8.	Розробка матеріалів графічної частини проекту	21.05	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	01.06	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до «Програми та методики тестування».

Пояснювальна записка
до дипломного проєкту

на тему: **Бібліотека для визначення контурів рельєфу з супутникових фотографій**

КПІ.ІТ-0321.045490.02.81

Київ – 2024

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

HiRISE – High Resolution Imaging Science Experiment – камера, встановлена на борту космічного апарату “Mars Reconnaissance Orbiter” для вивчення Марса.

BPMN – Business Process Model and Notation.

CLI – Command Line Interface – інтерфейс командного рядка.

PDS – Planetary Data System – архівна система для зберігання, організації та поширення даних з космічних місій NASA.

URL – Uniform Resource Locator – уніфікований локатор ресурсу.

HTTPS – HyperText Transfer Protocol Secure – розширення протоколу HTTP, яке використовується для забезпечення захищеного обміну інформацією.

ReLU – Rectified Linear Unit – функція активації, яка використовується в нейронних мережах.

GPU – Graphics Processing Unit – графічний процесор.

CUDA – Compute Unified Device Architecture – програмно-апаратна архітектура паралельних обчислень, розроблена NVIDIA для обчислень на графічних процесорах.

VRAM – Video Random Access Memory – тип пам’яті, спеціально призначеної для зберігання даних, необхідних для обробки та виведення графіки.

RAM – Random Access Memory – тип оперативної пам’яті комп’ютера, який використовується для тимчасового зберігання і швидкого доступу до даних та програм.

ПЗ – Програмне забезпечення.

ВСТУП

Сучасні наукові дослідження в космосі відкривають перед людством безмежні можливості для розуміння історії та природних процесів планет у нашій сонячній системі. Однією з ключових складових цього дослідницького потенціалу є аналіз супутникових зображень, зокрема фотографій, які дозволяють докладно вивчати рельєф та геологічні особливості планет.

Тема даної роботи, а саме розробка бібліотеки для визначення контурів рельєфу з супутникових фотографій, стає надзвичайно актуальною з наступних кількох причин.

Наукові дослідження планет, зокрема Марса, набувають все більшого значення. Збір та аналіз супутникових даних, зокрема високоякісних фотографій з камери HiRISE [1], відкривають нові перспективи для розуміння геологічних, кліматичних та геоморфологічних процесів на цих планетах.

Друга причина, що вирізняє актуальність даної роботи, полягає у стрімкому розвитку технологій обробки зображень і комп’ютерного зору (Computer Vision) [2]. Використання сучасних методів та мов програмування, таких як Python [3], дозволяє ефективно обробляти великі обсяги даних та отримувати точні результати.

Зростаючий інтерес до космічних досліджень, в тому числі досліджень Марса, свідчить про потребу в нових інструментах для аналізу супутникових даних. Дана робота може відігравати важливу роль у забезпеченні наукової спільноти, пропонуючи доступ до зручних та ефективних інструментів для аналізу рельєфу Марса та інших планет.

Практичні застосування даної роботи виходять далеко за межі наукових досліджень. Аналіз супутникових зображень рельєфу може бути використаний для планування майбутніх космічних місій, геологічних досліджень, а також у навчанні штучного інтелекту для аналізу та розпізнавання геологічних структур.

1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області

Аналіз рельєфу планет є важливою складовою космічних досліджень. У випадку Марса, визначення контурів рельєфу є ключовим для розуміння його геологічної історії, еволюції поверхні та виявлення потенційно цікавих об’єктів для подальших досліджень. Камера HiRISE надає високороздільні зображення Марса, які можуть бути використані для визначення контурів рельєфу.

Основним напрямком розробок є створення програмного забезпечення, яке здатне автоматично визначати контури рельєфу на супутникових зображеннях, отриманих з камери HiRISE. Це включає в себе розробку алгоритмів обробки зображень, які виявлять та відокремлять різні геологічні структури, такі як канали, кратери, льодовики тощо.

На сьогоднішній день, завдяки стрімкому розвитку обчислювальних технологій, використання штучного інтелекту, інструментів машинного навчання та глибокого навчання, стало можливим автоматизувати процес аналізу супутникових зображень. Методи обробки зображень, такі як виявлення контурів, почали широко використовуватись завдяки доступності потужних комп’ютерів та графічних процесорів.

Незважаючи на те, що наявні методи можуть автоматизувати аналіз зображень, процес аналізу великої кількості фотографій, знятих камерою HiRISE, все ще може бути часо- та працезатратним. Ручна обробка великої кількості фотографій може призвести до помилок через втомленість та людський фактор.

Розробка Python бібліотеки для автоматичного визначення контурів рельєфу з супутникових фотографій з камери HiRISE є актуальною і обґрунтованою. Використання комп’ютерів, обчислювальних технологій та методів штучного інтелекту дозволить автоматизувати процес аналізу даних та уникнути людських помилок, забезпечуючи більш точні результати.

1.2 Аналіз існуючих рішень

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації бібліотеки для визначення контурів рельєфу з супутникових фотографій. Далі будуть розглянуті готові програмні рішення, допоміжні програмні засоби та засоби розробки.

1.2.1 Аналіз відомих програмних продуктів

GDAL [4] – це бібліотека для роботи з географічними даними, яка підтримує багато різних форматів растрових та векторних даних. Вона надає засоби для читання, запису та обробки географічних даних, включаючи супутникові зображення. Хоча GDAL має певні можливості обробки зображень, вона в основному спрямована на роботу з географічними даними. GDAL доступний для використання в багатьох мовах програмування, включаючи Python, C++, Java та інші.

OpenCV [5] – це бібліотека для комп’ютерного зору та машинного навчання, яка містить величезний набір функцій для обробки зображень та відео. Вона надає інструменти для відкриття, обробки, аналізу та відображення зображень. OpenCV має багато алгоритмів для виявлення контурів, фільтрації зображень, виявлення границь та багато іншого. OpenCV доступний для використання в різних мовах програмування, включаючи Python, C++, Java та інші.

Scikit-image [6] – це бібліотека для обробки зображень на мові програмування Python, яка спеціалізується на обробці та аналізі зображень. Вона містить різнопланові функції для обробки зображень, такі як фільтрація, виявлення контурів, відображення та багато іншого. Scikit-image також підтримує роботу із растровими даними. Вона спеціалізується на Python і надає зручний інтерфейс для роботи з обробкою зображень у цій мові.

Для порівняння проєкту з аналогами можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогами

Функціонал	Дипломний проєкт	GDAL	OpenCV	scikit-image
Завантаження зображення з растрових даних	Так	Так	Частково (залежить від формату)	Ні
Визначення контурів рельєфу	Так	Ні	Так	Так
Фільтрація зображення	Так	Обмежено	Так	Так
Виявлення границь	Так	Ні	Так	Так
Задання параметрів для обробки зображення через CLI	Так	Ні	Так	Так
Відображення результатів	Так	Обмежено	Так	Так
Робота з географічними даними	Так	Так	Ні	Ні

1.2.2 Аналіз відомих алгоритмічних та технічних рішень

Існують два основних підходи обробки супутникових зображень для визначення контурів рельєфу:

а) статистичні методи обробки зображень: для цього підходу

використовуються алгоритми, які не потребують тренування, такі як реалізовані у бібліотеках OpenCV та scikit-image. Ці методи швидкі, але можуть бути чутливими до шуму та потребують постійної взаємодії з користувачем для налаштування певних параметрів обробки.

б) використання нейромережі UNet [7] для семантичної сегментації: цей підхід використовує нейронну мережу UNet, яка базується на зворотніх зв'язках та використовує параметри, попередньо натреновані на схожих даних. Нейромережі, такі як UNet, потребують попередніх тренувань, але в подальшому можуть працювати автономно та бути нечутливими до варіацій даних, звісно якщо обрані дані є достатньо схожими із тренувальним датасетом.

Для реалізації поставленої задачі в рамках цієї роботи, буде використано обидва підходи:

а) перший підхід (статистичні методи обробки зображень):

1) використовуватиметься для попередньої обробки даних: конвертації форматів, зменшення розширення (стискання), визначення грубих контурів.

2) даний підхід дозволить швидко й ефективно підготувати зображення для подальших процесів аналізу, зменшуючи їх розмір та виокремлюючи загальні грубі контури.

б) другий підхід (використання нейромережі UNet):

1) використовуватиметься для автоматичного визначення точних контурів на очищених (підготовлених першим підходом) зображеннях.

2) UNet надасть можливість більш точного та автономного визначення контурів рельєфу на попередньо підготовлених зображеннях.

Вказана комбінація дозволить ефективно використовувати переваги обох методів: швидкість та простоту першого для попередньої обробки даних і точність та автономність другого для визначення точних контурів рельєфу.

1.3 Опис бізнес-процесів

Для опису бізнес процесу обробки зображення використовується BPMN модель, яку наведено у графічному матеріалі, креслення 2.

Опис послідовності процесу обробки зображення:

– користувач відкриває термінал;

– користувач вписує команду для запуску та вказує ID необхідного зображення;

– якщо зображення не завантажене локально, тоді відбувається завантаження;

– якщо зображення вже завантажене, тоді виведеться відповідне повідомлення;

– користувач при запуску може також задати власні значення для доступних параметрів обробки;

– якщо ці параметри задані користувачем, тоді будуть використовуватись саме вказані ним значення;

– якщо параметри не були задані користувачем, будуть використовуватись прописані значення з config файлу;

– перевіряється чи існує стиснута версія в форматі PNG [8] оригінального зображення формату JP2 [9];

– якщо стиснутого зображення немає, тоді відбувається відповідний процес стискання;

– якщо стиснуте зображення вже існує, тоді виведеться відповідне повідомлення;

– відбувається процес розподілення зображення на плитки заданого розміру для подальшої передачі в UNet;

– відбувається процес генерації маски за допомогою UNet;

– відбувається процес визначення контурів за допомогою OpenCV;

– відбувається процес збереження масиву з координатами контуру у вигляді NPY [10] файлу та відповідної візуальної складової у вигляді PNG зображення;

– користувач має можливість переглянути результати.

1.4 Постановка задачі

Розробити набір методів автоматичного аналізу фотографій, об’єднаних в одну Python бібліотеку. Дана бібліотека має бути зручною у використанні і потребувати мінімальної взаємодії з боку дослідника. Процес обробки фотографій має бути швидким і ефективним для знімків, зроблених камерою HiRISE.

Висновки до розділу

В цьому розділі було виконано передпроєктне обстеження предметної області.

У підрозділі «Аналіз предметної області» було виявлено, що аналіз рельєфу планет, зокрема Марса, є ключовим для космічних досліджень, а камера HiRISE надає важливі дані для досягнення цієї мети. Високороздільні зображення, отримані цією камерою, стають ключовим ресурсом для визначення контурів рельєфу. Проте процес мануальної обробки цих зображень займає значний час, тому автоматизація цього процесу стає важливою задачею для отримання точних та ефективних для подальшого використання результатів.

У підрозділі «Аналіз існуючих рішень» проведено детальний аналіз різних програмних продуктів, таких як GDAL, OpenCV та scikit-image, та виявлено їх функціональність у контексті розробки бібліотеки. Зазначено, що кожен з цих інструментів має свої переваги та обмеження, які варто врахувати під час розробки бібліотеки для визначення контурів рельєфу.

У підрозділі «Опис бізнес-процесів» була розглянута послідовна схема бізнес-процесу обробки зображення, представлена у вигляді BPMN моделі. Було описано кожен крок процесу, враховуючи можливі сценарії.

У підрозділі «Постановка задачі» були сформульовані основні вимоги до розробленої бібліотеки. Зазначено, що зручність у використанні та швидкість обробки зображень є ключовими для досягнення успішних результатів в розробці бібліотеки для визначення контурів рельєфу з супутникових фотографій.

2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Варіанти використання програмного забезпечення

Головною функцією програмного забезпечення є визначення контурів рельєфу з супутникових фотографій, більше функцій можна побачити на діаграмі варіантів використання, яку наведено у графічному матеріалі, креслення 1.

В таблицях 2.1 - 2.15 наведені варіанти використання програмного забезпечення.

Таблиця 2.1 - Варіант використання UC-01

Use case name	Вибір зображення для обробки
Use case ID	UC-01
Goals	Вибір необхідної фотографії для подальшого знаходження контурів рельєфу
Actors	Користувач
Trigger	Користувач бажає вибрати необхідну фотографію
Pre-conditions	-
Flow of Events	Користувач запускає термінал та вписує ID необхідного зображення, використовуючи відповідний параметр після основної команди запуску
Extension	Якщо зображення вже завантажено, виведеться відповідне повідомлення та повторного завантаження не відбудеться. Якщо зображення із заданим ID немає, тоді відбудеться процес його завантаження, що також супроводжуватиметься відповідним повідомленням.
Post-Condition	Необхідне користувачу зображення успішно обране та готове для подальшої обробки

Таблиця 2.2 - Варіант використання UC-02

Use case name	Вибір директорії з даними
Use case ID	UC-02

Продовження таблиці 2.2

Goals	Вибрати директорію з необхідними для роботи даними
Actors	Користувач
Trigger	Користувач бажає вказати шлях до директорії з даними
Pre-conditions	-
Flow of Events	Користувач запускає термінал та вписує шлях до необхідної директорії із вхідними даними, використовуючи відповідний параметр після основної команди запуску
Extension	Якщо даний параметр не заданий користувачем, то буде використовуватись стандартне значення із файлу конфігурації. Якщо параметр був заданий користувачем, тоді значення із файлу конфігурації буде перезаписане.
Post-Condition	Вказану користувачем директорію з необхідними для роботи даними успішно вибрано

Таблиця 2.3 - Варіант використання UC-03

Use case name	Вказання шляху до файлу конфігурації
Use case ID	UC-03
Goals	Вказати шлях до конфігураційного файлу, що містить параметри для обробки зображення
Actors	Користувач
Trigger	Користувач бажає вказати шлях до файлу конфігурації
Pre-conditions	-
Flow of Events	Користувач запускає термінал та вписує шлях до необхідного йому файлу конфігурації, використовуючи відповідний параметр після основної команди запуску
Extension	Якщо даний параметр не заданий користувачем, то буде використовуватись значення, вказаний за замовчуванням
Post-Condition	Шлях до необхідного файлу конфігурації успішно вказано

Таблиця 2.4 - Варіант використання UC-04

Use case name	Вказання шляху до натренованої моделі UNet

Продовження таблиці 2.4

Use case ID	UC-04
Goals	Вказати шлях до директорії, де знаходиться модель UNet
Actors	Користувач
Trigger	Користувач бажає вказати шлях до необхідної моделі UNet
Pre-conditions	-
Flow of Events	Користувач запускає термінал та вписує шлях до необхідної директорії із натренованою моделлю UNet, використовуючи відповідний параметр після основної команди запуску
Extension	Якщо даний параметр не заданий користувачем, то буде використовуватись стандартне значення із файлу конфігурації. Якщо параметр був заданий користувачем, тоді значення із файлу конфігурації буде перезаписане.
Post-Condition	Вказану користувачем директорію із натренованою моделлю UNet успішно вибрано

Таблиця 2.5 - Варіант використання UC-05

Use case name	Вибір одного з каналів у зображенні
Use case ID	UC-05
Goals	Вибрати один з каналів у зображенні для подальшої обробки
Actors	Користувач
Trigger	Користувач бажає вибрати певний канал у зображенні для подальшої обробки
Pre-conditions	Користувач має знати скільки каналів містить обране зображення
Flow of Events	Користувач запускає термінал та вказує номер необхідного каналу у зображенні на обробку, використовуючи відповідний параметр після основної команди запуску
Extension	Якщо даний параметр не заданий користувачем, то буде використовуватись стандартне значення із файлу конфігурації. Якщо параметр був заданий користувачем, тоді значення із файлу конфігурації буде перезаписане.
Post-Condition	Необхідний канал у зображенні успішно обрано для виконання подальшої обробки

Таблиця 2.6 - Варіант використання UC-06

Use case name	Вказання коефіцієнту стискання
Use case ID	UC-06
Goals	Вказати коефіцієнт стискання для переведення зображення з формату JP2 до формату PNG
Actors	Користувач
Trigger	Користувач бажає вказати коефіцієнт стискання
Pre-conditions	-
Flow of Events	Користувач запускає термінал та вказує значення для коефіцієнту стиснення зображень, використовуючи відповідний параметр після основної команди запуску
Extension	Якщо даний параметр не заданий користувачем, то буде використовуватись стандартне значення із файлу конфігурації. Якщо параметр був заданий користувачем, тоді значення із файлу конфігурації буде перезаписане.
Post-Condition	Вказаний користувачем коефіцієнт стискання для фотографій успішно встановлено

Таблиця 2.7 - Варіант використання UC-07

Use case name	Вказання розміру плиток, на які ділиться зображення
Use case ID	UC-07
Goals	Вказати розмірність плиток, на які ділиться зображення під час проходження обробки
Actors	Користувач
Trigger	Користувач бажає задати розмірність плиток
Pre-conditions	-
Flow of Events	Користувач запускає термінал та вказує значення розміру плиток, використовуючи відповідний параметр після основної команди запуску
Extension	Якщо даний параметр не заданий користувачем, то буде використовуватись стандартне значення із файлу конфігурації. Якщо параметр був заданий користувачем, тоді значення із файлу конфігурації буде перезаписане.

Продовження таблиці 2.7

Post-Condition	Заданий користувачем розмір плиток, на які буде розподілятись зображення, успішно встановлено

Таблиця 2.8 - Варіант використання UC-08

Use case name	Увімкнення відображення метаданих
Use case ID	UC-08
Goals	Увімкнути відображення метаданих до обраної фотографії
Actors	Користувач
Trigger	Користувач бажає увімкнути відображення метаданих
Pre-conditions	-
Flow of Events	Користувач запускає термінал і вводить основну команду запуску, включаючи відповідний параметр-флаг для виводу та відображення метаданих фотографії
Extension	Якщо даний параметр не заданий користувачем, то відображення метаданих не буде виконане за замовчуванням. Якщо параметр був прописаний користувачем, тоді буде здійснено вивід-відображення відповідних до фотографії метаданих.
Post-Condition	У процесі обробки фотографії буде відображено відповідні метадані.

Таблиця 2.9 - Варіант використання UC-09

Use case name	Вказання загальної кількості каналів у зображенні
Use case ID	UC-09
Goals	Вказати загальну кількість каналів у зображенні
Actors	Користувач
Trigger	Користувач бажає вказати загальну кількість каналів у зображенні
Pre-conditions	-
Flow of Events	Користувач запускає термінал та вказує значення кількості каналів у зображенні, використовуючи відповідний параметр після основної команди запуску

Продовження таблиці 2.9

Extension	Якщо даний параметр не заданий користувачем, то буде використовуватись стандартне значення із файлу конфігурації. Якщо параметр був заданий користувачем, тоді значення із файлу конфігурації буде перезаписане.
Post-Condition	Вказане користувачем значення кількості каналів у зображенні успішно встановлено

Таблиця 2.10 - Варіант використання UC-10

Use case name	Вказання кількості семантичних класів, які потрібно виявити на зображенні
Use case ID	UC-10
Goals	Вказати кількість семантичних класів, які необхідно виявити на зображенні
Actors	Користувач
Trigger	Користувач бажає вказати кількість семантичних класів, які необхідно виявити на зображенні
Pre-conditions	-
Flow of Events	Користувач запускає термінал та вказує значення кількості семантичних класів, які необхідно виявити на зображенні, використовуючи відповідний параметр після основної команди запуску
Extension	Якщо даний параметр не заданий користувачем, то буде використовуватись стандартне значення із файлу конфігурації. Якщо параметр був заданий користувачем, тоді значення із файлу конфігурації буде перезаписане.
Post-Condition	Вказане користувачем значення кількість семантичних класів, які необхідно виявити на зображенні, успішно встановлено

Таблиця 2.11 - Варіант використання UC-11

Use case name	Встановлення порогу чутливості для конвертації матриці ймовірностей у бінарну маску
Use case ID	UC-11
Goals	Встановити значення порогу чутливості для конвертації матриці ймовірностей у бінарну маску

Продовження таблиці 2.11

Actors	Користувач
Trigger	Користувач бажає встановити значення порогу чутливості для конвертації матриці ймовірностей у бінарну маску
Pre-conditions	-
Flow of Events	Користувач запускає термінал та вказує значення порогу чутливості, використовуючи відповідний параметр після основної команди запуску
Extension	Якщо даний параметр не заданий користувачем, то буде використовуватись стандартне значення із файлу конфігурації. Якщо параметр був заданий користувачем, тоді значення із файлу конфігурації буде перезаписане.
Post-Condition	Вказане користувачем значення порогу чутливості успішно встановлено

Таблиця 2.12 - Варіант використання UC-12

Use case name	Встановлення значення для заповнення прогалин
Use case ID	UC-12
Goals	Встановити значення розміру в пікселях, нижче якого всі прогалини будуть заповнені
Actors	Користувач
Trigger	Користувач бажає встановити значення для заповнення прогалин
Pre-conditions	-
Flow of Events	Користувач запускає термінал та вказує значення для заповнення прогалин, використовуючи відповідний параметр після основної команди запуску
Extension	Якщо даний параметр не заданий користувачем, то буде використовуватись стандартне значення із файлу конфігурації. Якщо параметр був заданий користувачем, тоді значення із файлу конфігурації буде перезаписане.
Post-Condition	Вказане користувачем значення для заповнення прогалин успішно встановлено

Таблиця 2.13 - Варіант використання UC-13

Use case name	Встановлення значення для видалення острівців
Use case ID	UC-13
Goals	Встановити значення розміру в пікселях, нижче якого всі острівці будуть видалені
Actors	Користувач
Trigger	Користувач бажає встановити значення для видалення острівців
Pre-conditions	-
Flow of Events	Користувач запускає термінал та вказує значення для видалення острівців, використовуючи відповідний параметр після основної команди запуску
Extension	Якщо даний параметр не заданий користувачем, то буде використовуватись стандартне значення із файлу конфігурації. Якщо параметр був заданий користувачем, тоді значення із файлу конфігурації буде перезаписане.
Post-Condition	Вказане користувачем значення для видалення острівців успішно встановлено

Таблиця 2.14 - Варіант використання UC-14

Use case name	Встановлення значення відстані між точками, які формують контур
Use case ID	UC-14
Goals	Встановити значення відстані в пікселях між точками, які формують визначений контур
Actors	Користувач
Trigger	Користувач бажає вказати значення відстані між точками, які формують контур
Pre-conditions	-
Flow of Events	Користувач запускає термінал та вказує значення відстані в пікселях між точками, які формуватимуть контур, використовуючи відповідний параметр після основної команди запуску

Продовження таблиці 2.14

Extension	Якщо даний параметр не заданий користувачем, то буде використовуватись стандартне значення із файлу конфігурації. Якщо параметр був заданий користувачем, тоді значення із файлу конфігурації буде перезаписане.
Post-Condition	Вказане користувачем значення к значення відстані між точками успішно встановлено

Таблиця 2.15 - Варіант використання UC-15

Use case name	Перегляд результатів
Use case ID	UC-15
Goals	Переглянути результати обробки для знаходження контурів рельєфу
Actors	Користувач
Trigger	Користувач бажає переглянути отримані результати з визначення контурів
Pre-conditions	-
Flow of Events	Користувач переходить у директорію з даними, де може відкрити результати обробки у вигляді зображення визначеного контуру та файлу з масивом його координат
Extension	-
Post-Condition	Користувач переглядає зображення визначеного контуру та масив його координат

2.2 Аналіз системних вимог

Для ефективної роботи з бібліотекою необхідні:

– наявність GPU NVIDIA з підтримкою CUDA та мінімальним обсягом 4 ГБ VRAM. Це дозволить використовувати швидкість обчислень на GPU для виконання складних алгоритмів обробки зображень та аналізу контурів рельєфу. Мінімальний обсяг 4 ГБ VRAM дозволить ефективно опрацьовувати великі зображення високої роздільної здатності;

– достатній об’єм фізичного накопичувача. Зображення високої роздільної здатності можуть займати значний об’єм місця на диску, тому важливо мати достатньо вільного місця для їх зберігання та обробки;

– мінімум 8 ГБ RAM. Це дозволить запускати обчислення та обробку зображень у великій кількості, забезпечуючи при цьому високу продуктивність та швидкість роботи.

2.3 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. В таблиці 2.16 наведено загальну модель вимог, а в таблиці 2.17 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 2.1.

Таблиця 2.16 – Загальна модель вимог

№	Назва	ID вимоги	Пріоритети	Ризики
1	Використання CLI	REQ-1	Високий	Високий
1.1	Вибір зображення для обробки.	REQ-2	Високий	Високий
1.1.1	Перевірка на правильність введеного типу параметра	REQ-3	Високий	Низький
1.1.2	Перевірка наявності завантаженого зображення.	REQ-4	Високий	Високий
1.1.3	Завантаження зображення.	REQ-5	Високий	Високий
1.2	Вибір директорії з даними.	REQ-6	Високий	Низький
1.2.1	Вказання шляху до директорії з даними.	REQ-7	Середній	Низький
1.2.2	Перевірка на правильність введеного типу параметра	REQ-8	Високий	Низький

Продовження таблиці 2.16

№	Назва	ID вимоги	Пріоритети	Ризики
1.3	Вибір файлу конфігурації.	REQ-9	Високий	Середній
1.3.1	Вказання шляху до файлу конфігурації.	REQ-10	Середній	Низький
1.3.2	Перевірка на правильність введеного типу параметра	REQ-11	Високий	Низький
1.4	Вибір моделі UNet	REQ-12	Високий	Високий
1.4.1	Вказання шляху до натренованої моделі UNet	REQ-13	Середній	Низький
1.4.2	Перевірка на правильність введеного типу параметра	REQ-14	Високий	Низький
1.5	Вибір одного з каналів у зображенні	REQ-15	Середній	Середній
1.5.1	Перевірка на правильність введеного типу параметра	REQ-16	Високий	Низький
1.6	Вказання коефіцієнту стискання	REQ-17	Середній	Низький
1.6.1	Перевірка на правильність введеного типу параметра	REQ-18	Високий	Низький
1.7	Вказання розміру плиток, на які ділиться зображення	REQ-19	Середній	Середній
1.7.1	Перевірка на правильність введеного типу параметра	REQ-20	Високий	Низький
1.8	Увімкнення відображення метаданих	REQ-21	Середній	Низький
1.9	Вказання загальної кількості каналів у зображенні	REQ-22	Середній	Середній

Продовження таблиці 2.16

№	Назва	ID вимоги	Пріоритети	Ризики
1.9.1	Перевірка на правильність введеного типу параметра	REQ-23	Високий	Низький
1.10	Вказання кількості семантичних класів, які потрібно виявити на зображенні	REQ-24	Середній	Середній
1.10.1	Перевірка на правильність введеного типу параметра	REQ-25	Високий	Низький
1.11	Встановлення порогу чутливості для конвертації матриці ймовірностей у бінарну маску	REQ-26	Високий	Середній
1.11.1	Перевірка на правильність введеного типу параметра	REQ-27	Високий	Низький
1.12	Встановлення значення для заповнення прогалин	REQ-28	Високий	Середній
1.12.1	Перевірка на правильність введеного типу параметра	REQ-29	Високий	Низький
1.13	Встановлення значення для видалення острівців	REQ-30	Високий	Середній
1.13.1	Перевірка на правильність введеного типу параметра	REQ-31	Високий	Низький
1.14	Встановлення значення відстані між точками, які формують контур	REQ-32	Високий	Середній
1.14.1	Перевірка на правильність введеного типу параметра	REQ-33	Високий	Низький

Продовження таблиці 2.16

№	Назва	ID вимоги	Пріоритети	Ризики
2	Підготовка зображення для передачі на модель UNet	REQ-34	Високий	Високий
2.1	Зчитування зображення	REQ-35	Високий	Високий
2.2	Перевірка наявності стиснутої версії наданого зображення	REQ-36	Високий	Низький
2.3	Перевірка кратності розширення зображення на розмір плитки	REQ-37	Високий	Високий
2.4	Розподілення зображення на плитки	REQ-38	Середній	Середній
3	Аналіз за допомогою UNet	REQ-39	Високий	Середній
3.1	Перевірка плиток на наявність необхідних даних	REQ-40	Середній	Низький
3.2	Генерації семантичної маски для кожної окремої плитки	REQ-41	Високий	Середній
3.3	Відтворення загальної маски з окремих фрагментів	REQ-42	Середній	Середній
4	Визначення контуру	REQ-43	Високий	Високий
4.1	Визначення зовнішньої границі пікселів навколо бінарної маски	REQ-44	Середній	Середній
4.2	Розрахунок поліному, який описує знайдений контур	REQ-45	Високий	Середній
5	Збереження результатів	REQ-46	Високий	Низький

Продовження таблиці 2.16

№	Назва	ID вимоги	Пріоритети	Ризики
5.1	Збереження списку координат у вигляді NPY файлу	REQ-47	Високий	Низький
5.2	Збереження графічного представлення контуру	REQ-48	Високий	Низький

Таблиця 2.17 – Перелік функціональних вимог

ID вимоги	Назва та опис
FR-01	Вибір зображення для обробки Система має надавати можливість вибирати необхідні супутникові зображення для подальшої їх передачі на обробку.
FR-02	Вибір директорії з даними Система має надавати можливість обрати необхідну директорію з даними.
FR-03	Вибір конфігураційного файлу Система має надавати можливість, за потреби, вибрати той чи інший конфігураційний файл, що необхідний для проведення обробки зображень.
FR-04	Вибір натренованої моделі UNet Система має надавати можливість, за потреби, вибрати ту чи іншу натреновану модель нейромережі UNet.
FR-05	Вибір певного каналу у зображенні Система має надавати можливість обрати необхідний канал у зображені, що підлягатиме подальшій обробці для отримання контурів рельєфу.

Продовження таблиці 2.17

ID вимоги	Назва та опис
FR-06	Вибір коефіцієнту стискання зображення Система має надавати можливість вказати певний коефіцієнт стискання зображення для отримання різнопланових результатів.
FR-07	Вибір розміру плиток, на які розподіляється зображення Система має надавати можливість встановити розмір плиток, на які розподіляється зображення під час обробки.
FR-08	Відображення метаданих Система має надавати можливість переглядати метадані до наданих супутникових фотографій.
FR-09	Вказання загальної кількості каналів у зображенні Система має надавати можливість, за потреби, вказати загальну кількість каналів у вхідному зображенні.
FR-10	Вказання кількості семантичних класів Система має надавати можливість вказати кількість семантичних класів, які необхідно виявити на зображенні.
FR-11	Встановлення порогу чутливості Система має надавати можливість, за потреби, встановити значення порогу чутливості для конвертації матриці ймовірностей у бінарну маску.
FR-12	Встановлення значення для заповнення прогалин Система має надавати можливість встановити значення розміру прогалини, менше якого всі прогалини будуть заповнюватись пікселями.
FR-13	Встановлення значення для видалення острівців Система має надавати можливість встановити значення розміру острівця, менше якого всі острівці будуть видалені.

Продовження таблиці 2.17

ID вимоги	Назва та опис
FR-14	Встановлення відстані між точками, які формують контур Система має надавати можливість встановити значення відстані між точками, які формують визначений контур рельєфу.
FR-15	Отримання результатів у вигляді списку координат контуру Система має надавати можливість отримати результат обробки фотографії у вигляді списку координат, через які проходить контур рельєфу.
FR-16	Отримання результатів у графічному вигляді Система має надавати можливість отримати результат обробки фотографії у вигляді зображення з графічно визначеним контуром відповідного рельєфу.

	FR-01	FR-02	FR-03	FR-04	FR-05	FR-06	FR-07	FR-08	FR-09	FR-10	FR-11	FR-12	FR-13	FR-14	FR-15	FR-16
UC-01	+															
UC-02		+														
UC-03			+													
UC-04				+												
UC-05					+											
UC-06						+										
UC-07							+									
UC-08								+								
UC-09									+							
UC-10										+						
UC-11											+					
UC-12												+				
UC-13													+			
UC-14														+		
UC-15															+	+

Рисунок 2.1 – Матриця трасування вимог

2.4 Розроблення нефункціональних вимог

У даному продукті мають бути дотримані такі нефункціональні вимоги:

– продуктивність: бібліотека повинна бути ефективною та швидкою в обробці супутникових зображень для визначення контурів рельєфу;

– масштабованість: система повинна бути спроможною працювати з різними розмірами та роздільними здатностями супутникових зображень, забезпечуючи при цьому стабільну продуктивність;

– точність результатів: бібліотека повинна надавати точні та надійні результати визначення контурів рельєфу;

– модульність та розширюваність: система повинна мати модульну архітектуру, що дозволяє легко розширювати функціональність та додавати нові можливості у майбутньому;

– стабільність та надійність: бібліотека повинна бути стабільною та надійною у роботі, мінімізуючи виникнення помилок та аварійних ситуацій;

– легкість використання: інтерфейс та документація бібліотеки повинні бути зрозумілими та зручними для використання;

– керованість параметрами аналізу: система повинна надавати можливість користувачеві налаштовувати параметри аналізу зображень.

Висновки до розділу

В цьому розділі було виконано розробку вимог до програмного забезпечення.

У підрозділі «Варіанти використання програмного забезпечення» у вигляді Use-case діаграми були представлені відповідні варіанти використання. Надалі кожен із них був описаний в окремих таблицях.

У підрозділі «Аналіз системних вимог» було зазначено важливі характеристики обчислювального пристрою для коректної та ефективної роботи із програмним продуктом.

У підрозділі «Розроблення функціональних вимог» було наведено загальну модель вимог та список функціональних вимог у вигляді відповідних таблиць. На основі зв’язків між варіантами використання та встановленими функціональними вимогами було побудовано матрицю трасування.

У підрозділі «Розроблення нефункціональних вимог» було розглянуто список нефункціональних вимог, що мають бути дотримані для стабільної та зручної роботи зі створюваною бібліотекою.

За результатами розділу сформовано технічне завдання на розробку програмного забезпечення.

3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмного забезпечення

Архітектура бібліотеки представлена у вигляді діаграми пакетів, яку наведено у графічному матеріалі, плакат «Діаграма пакетів бібліотеки».

Дана структура пакетів включає в себе:

а) Configs – папка для файлів конфігурації, яка містить:

1) config_parser.py – клас для обробки аргументів за замовчуванням із config.yml;

2) config.yml – визначення стандартних аргументів.

б) Core – папка, що містить основні функції та методи, а саме:

1) JP2tools.py – колекція класів, які працюють з оригінальними зображеннями формату JP2;

2) PngTools.py – колекція класів, які працюють зі стиснутими зображеннями формату PNG;

3) utility.py – колекція допоміжних функцій.

в) EnvConfigs – папка для файлів конфігурації середовища, яка містить:

1) contours.yml – файл конфігурації для середовища conda; встановити, запустивши “conda env create -f contours.yml”.

г) UNet – папка з реалізацією PyTorch UNet, яка використовується для розрахунку масок сегментації плиток, на які розподілене зображення, розміром 512x512 пікселів. Вона включає:

1) weights – папка для збереження натренованої моделі нейромережі UNet;

2) model.py – архітектура нейронної мережі.

д) main.py – основний файл, який об’єднує функціональні можливості всіх інших модулів у ланцюжок обробки зображень;

е) .gitignore – файл, що містить розширення/папки файлів, які

потрібно ігнорувати під час синхронізації з GitHub;

ж) README.md – файл із набором інструкцій з використання.

3.2 Обґрунтування вибору засобів розробки

Python було обрано як основну мову програмування через свою широку популярність та велику кількість наукових бібліотек, зокрема для аналізу зображень. Python – це мова високого рівня, що спрощує розробку та надає багато можливостей для обробки та аналізу даних. Також, наявність фреймворків для машинного навчання, таких як PyTorch [11], сприяє використанню складних моделей, наприклад, UNet, для сегментації зображень.

У порівнянні з іншими мовами програмування, такими як C++ або Java, Python відзначається більшою простотою та лаконічністю коду, що сприяє швидкому розвитку та зрозумілості програмного забезпечення.

Conda [12] було обрано як інструмент для управління віртуальним середовищем Python через його ефективний менеджмент та можливість швидко створювати та налаштовувати середовища для різних проєктів. Це дозволяє уникнути конфліктів між версіями бібліотек та забезпечити стабільність у роботі проєкту.

Порівнюючи з іншими інструментами для керування віртуальними середовищами, такими як virtualenv [13] або pipenv [14], Conda відзначається більшою гнучкістю та розширеними можливостями. Даний інструмент має широкий спектр доступних пакетів і версій, а також дозволяє створювати і керувати віртуальними середовищами. Це надає змогу легко і ефективно управляти залежностями проєкту, що робить Conda дуже зручним інструментом для розробки, особливо маючи справу із великою кількістю залежностей.

Visual Studio Code (VSCode) [15] був обраний як основне середовище розробки через свою популярність, широкі можливості розширення та інтеграцію з іншими інструментами. VSCode забезпечує зручний інтерфейс

для редагування коду, налаштування робочого середовища та відладки програмного забезпечення, що сприяє ефективному розвитку проєкту.

Порівнюючи з іншими середовищами розробки, такими як PyCharm чи Jupyter Notebook, VSCode відзначається більшою швидкістю та легкістю використання, а також можливістю розширення функціональності за допомогою різноманітних плагінів.

3.3 Конструювання програмного забезпечення

Для задачі визначення контурів рельєфу з супутникових фотографій було використано алгоритм UNet, що відомий завдяки своїй здатності до точної сегментації зображень. UNet складається з таких основних частин: encoder, bottleneck, decoder, skip connections та output.

Encoder складається з послідовності згорткових шарів (convolutional layers), кожен з яких застосовує фільтри для вилучення ознак зображення. Кожен рівень складається з двох згорткових шарів, кожен з яких використовує невеликі фільтри (зазвичай розміром 3x3). Після кожного згорткового шару застосовується функція активації ReLU (Rectified Linear Unit) [16], яка допомагає нейронній мережі навчати нелінійні характеристики. Після двох згорткових шарів виконується операція max pooling (зазвичай розміром 2x2) для зменшення розміру просторового виміру зображення, залишаючи найбільш значущі ознаки. Це дозволяє зменшити обчислювальні витрати та зосередити увагу на важливих деталях.

Після проходження через кілька рівнів encoder, зображення досягає найбільш стиснутого представлення в мережі, що називається bottleneck. Це місце, де зображення максимально зменшене в розмірах, але містить найбільш важливу інформацію. У bottleneck, як і в інших частинах мережі, використовуються згорткові шари з функціями активації ReLU для подальшого вилучення ознак.

Decoder відновлює просторовий розмір зображення, поступово збільшуючи його до оригінального розміру. Кожен рівень decoder містить

операцію up-convolution (транспонована згортка), яка збільшує розмір зображення. Після кожної up-convolution виконуються два згорткових шари з функцією активації ReLU, аналогічні до тих, що використовуються в encoder. На кожному рівні decoder вихід з відповідного рівня encoder з'єднується з поточним рівнем через skip connections, що дозволяє мережі використовувати як локальні, так і глобальні ознаки для більш точного відновлення зображення.

Skip connections з'єднують кожен рівень encoder з відповідним рівнем decoder. Це допомагає передати детальну інформацію, яка може бути втрачена під час операцій max pooling в encoder, безпосередньо до відповідних рівнів у decoder. Такий підхід покращує точність сегментації, оскільки поєднує низькорівневі (детальні) та високорівневі (абстрактні) ознаки.

Останній шар UNet (output) виконує фінальну згортку, щоб зменшити кількість каналів до необхідного числа класів сегментації, створюючи бінарну маску. Бінарна маска – це зображення, де кожен піксель належить або до класу «фон», або до класу «об'єкт». Це дозволяє точно визначити контури та структури на зображенні, які відповідають об'єктам інтересу, таким як контури рельєфу семантичних класів на супутникових фотографіях.

В рамках даної роботи, модель нейронної мережі UNet було натреновано на датасеті з супутникових фотографій з камери HiRISE для визначення контурів рельєфу полярних льодовикових мас на Марсі.

В даній роботі також використовується два алгоритми морфологічної обробки зображень з бібліотеки scikit-image:

– remove_small_holes [17] – цей алгоритм використовується для видалення маленьких дірок (білих областей) у бінарному зображенні, які мають площу меншу за вказаний поріг. Це дозволяє «заповнювати» невеликі прогалини всередині об'єктів, що є корисним для покращення якості сегментації;

– remove_small_objects [18] – цей алгоритм видаляє маленькі острівці (чорні області) у бінарному зображенні, які мають площу меншу за

вказаний поріг. Це дозволяє «очищати» зображення від дрібних об’єктів або шуму, які можуть бути артефактами сегментації.

Ці два методи застосовуються до бінарної маски, отриманої після сегментації зображення за допомогою моделі UNet. Їх метою є покращення якості сегментації шляхом видалення невеликих шумових елементів та заповнення дрібних прогалин, які могли залишитися після початкової обробки.

Останнім кроком в процесі обробки зображення є знаходження безпосередньо контурів рельєфу. Для цього використовується інструмент з бібліотеки OpenCV – алгоритм cv.findContours [19].

Алгоритм cv.findContours використовується для виявлення контурів на бінарній масці, яка отримана після сегментації зображення. Він проходить по всьому зображенню, знаходить початкову точку контуру і відстежує його до повернення в початкову точку. Цей процес повторюється для всіх контурів у зображенні. В даній роботі були використані наступні методи цього алгоритму:

– cv.RETR_EXTERNAL – це метод виявлення контурів, який вказує, що потрібно знаходити лише зовнішні контури, а не внутрішні;

– cv.CHAIN_APPROX_NONE – це метод, який вказує, що контури повинні бути повністю збережені, без апроксимації.

В якості сховища для вхідних зображень, проміжних виходів і результатів використовується локально створювана директорія “data”, структура якої представлена у вигляді діаграми, яку наведено у графічному матеріалі, плакат «Діаграма структури сховища».

Дана структура включає в себе:

а) images – папка, що зберігає зображення та складається з:

1) JP2 – папка, що містить оригінальні супутникові фотографії з камери HiRISE в форматі JP2;

2) png – папка для збереження стиснутих варіантів супутникових фотографій в форматі PNG.

б) results – папка для збереження результатів обробки фотографій, що складається з:

1) npy – папка для збереження списків координат визначених контурів у форматі NPY;

2) plots – папка для збереження зображень з графічно визначеними контурами.

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 3.1.

Таблиця 3.1 – Опис утиліт

№ п/п	Назва	Опис застосування
1	NumPy	Бібліотека для роботи з масивами даних, використовується для обчислення розмірів зображення.
2	OSGeo.GDAL	Основна бібліотека для роботи з геопросторовими даними, використовується для обробки растрових зображень формату JP2.
3	PyTorch	Бібліотека для роботи з нейронними мережами, використовується для визначення моделі UNet та обробки зображень.
4	OpenCV	Бібліотека для Computer Vision та обробки зображень. Використовується для знаходження контурів на бінарній масці.

Продовження таблиці 3.1

№ п/п	Назва	Опис застосування
5	scikit-image	З бібліотеки scikit-image було використано модуль skimage.morphology, який містить функції для морфологічної обробки зображень, таких як видалення малих об’єктів (острівців) та прогалин на бінарній масці зображення після сегментації.
6	os	Модуль в стандартній бібліотеці Python, який надає функції для взаємодії з операційною системою, зокрема для створення каталогів.
7	urllib.request	Цей модуль є частиною стандартної бібліотеки Python і використовується для взаємодії з різними ресурсами через мережу Інтернет. Використовується для завантаження JP2 зображень з Інтернету з використанням URL-адреси та зберігання їх локально на комп’ютері.
8	matplotlib.pyplot	Pyplot це частина бібліотеки Matplotlib для створення графіків та візуалізації даних у Python. Використовується для відображення зображень, створення графіків контурів об’єктів та збереження результатів обробки зображень у зручний формат.
9	scipy.interpolate	Модуль для інтерполяції сплайнами. За допомогою splprep і splev забезпечується побудова гладких контурів для результатів сегментації.
10	Conda	Інструмент для управління віртуальним середовищем Python.

Продовження таблиці 3.1

№ п/п	Назва	Опис застосування
11	Visual Studio Code	Головне середовище розробки програмного забезпечення.

Тексти програмного коду наведені в окремому документі «Текст програми».

3.4 Аналіз безпеки даних

Аспекти безпеки даних, що реалізовані в цій роботі:

а) завантаження даних з Інтернету:

1) перевірка наявності файлу: перед завантаженням JP2 файлу з сервера PDS (Planetary Data System) [20] виконується перевірка, чи існує вже файл зображення на локальному диску. Це зменшує кількість непотрібних завантажень і допомагає уникнути дублювання даних.

2) використання HTTPS: URL для завантаження використовує HTTPS, що забезпечує безпечну передачу даних через мережу, захищаючи їх від перехоплення.

б) зберігання даних:

1) директорії: створення директорій для зберігання JP2 файлів, PNG файлів та результатів обробки організує дані і зменшує ризик втрати або пошкодження файлів.

2) формат зберігання: координати контурів зберігаються у форматі NPY, який є оптимізованим для зберігання та обробки числових даних у мові Python.

Висновки до розділу

В цьому розділі було описано основні архітектурні та алгоритмічні рішення для виконуваної роботи.

У підрозділі «Архітектура програмного забезпечення» було розглянуто архітектуру розроблюваної бібліотеки, що представлена у вигляді діаграми структури папок і файлів. Також було наведено опис для кожного відповідного компоненту.

У підрозділі «Конструювання програмного забезпечення» було описано основні алгоритми та методи, що відповідають за вирішення поставлених задач та ефективність загальної роботи. Було описано використовуване сховище даних, що представлене у вигляді діаграми структури папок. Для кожної папки було наведено опис із вказанням призначення. Також було наведено таблицю із описом утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовувалось у розробці.

У підрозділі «Аналіз безпеки даних» було описано основні аспекти безпеки, що реалізовані у бібліотеці для визначення контурів рельєфу з супутникових фотографій.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості ПЗ

Метою тестування є наступне:

– перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;

– перевірка збереження даних;

– знаходження проблем, помилок і недоліків з метою їх усунення;

– перевірка зручності інтерфейсу командного рядка.

Метриками для оцінки якості ПЗ обрано наступні:

– кількість багів (помилок) у коді, що можуть спричинити некоректну роботу програмного забезпечення;

– кількість фрагментів коду, що не відповідають стандартам якісного написання коду (читабельність, функціональність тощо);

– кількість дублікатів у коді.

Для оцінки якості ПЗ за вищевказаними та іншими метриками було використано статичний аналізатор SonarCloud [21], що може аналізувати код без його виконання, перевіряючи наявність помилок, вразливостей тощо. Результати аналізу SonarCloud представлені на рисунку 4.1.

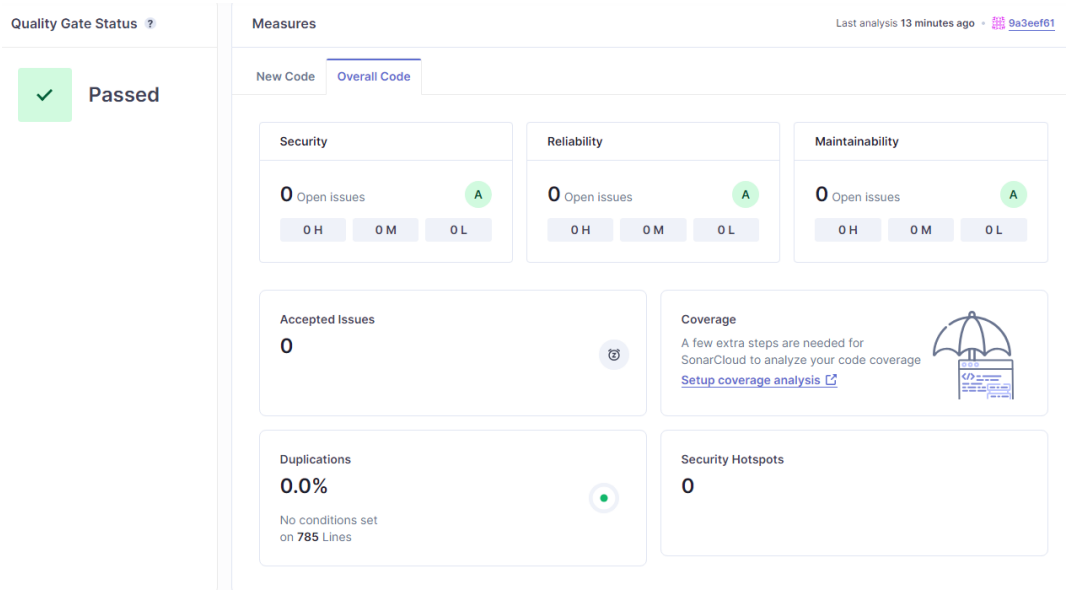


Рисунок 4.1 – Результати аналізу SonarCloud

З отриманих результатів можна зробити висновок, що помилок та вразливостей в коді не виявлено.

Для проведення аналізу продуктивності програмного забезпечення було використано динамічний аналізатор Py-Spy [22], який дозволяє профілювати виконання програм на Python в реальному часі. Приклад отриманих результатів аналізу представлено на рисунку 4.2.

```
(contours) PS C:\Users\Nikita\Desktop\diploma\ReliefContours> py-spy top -- python main.py --imID ESP_018244_2655

Collecting samples from '"python" main.py --imID ESP_018244_2655' (python v3.11.9)
Total Samples 1000
GIL: 21.00%, Active: 100.00%, Threads: 1

  %Own   %Total  OwnTime  TotalTime  Function (filename)
  0.00%   0.00%   1.64s     1.64s   TranslateInternal (osgeo\gdal.py)
  0.00%  30.00%   0.600s    2.41s   segment (PngTools.py)
  0.00%   0.00%   0.550s    0.550s  _conv_forward (torch\nn\modules\conv.py)
  0.00%   0.00%   0.490s    0.490s  ComputeRasterMinMax (osgeo\gdal.py)
  0.00%   0.00%   0.380s    1.27s   <module> (torch\__init__.py)
 32.00%  32.00%   0.320s    0.320s  _resample (matplotlib\image.py)
  0.00%   0.00%   0.290s    0.290s  batch_norm (torch\nn\functional.py)
  0.00%   0.00%   0.230s    0.230s  _path_stat (<frozen importlib._bootstrap_external>)
 15.00%  15.00%   0.220s    0.220s  label (scipy\ndimage\_measurements.py)
  0.00%   0.00%   0.200s    0.210s  _compile_bytecode (<frozen importlib._bootstrap_external>)
  2.00%   2.00%   0.170s    2.07s   _call_with_frames_removed (<frozen importlib._bootstrap>)
  0.00%   0.00%   0.170s    0.720s  forward (torch\nn\modules\conv.py)
  0.00%   0.00%   0.150s    0.150s  get_data (<frozen importlib._bootstrap_external>)
  7.00%   7.00%   0.140s    0.160s  load (PIL\ImageFile.py)
  0.00%   0.00%   0.090s    0.100s  __getattr__ (torch\_ops.py)
  0.00%   0.00%   0.090s    0.090s  realpath (<frozen ntpath>)
  4.00%  11.00%   0.090s    0.260s  _pil_png_to_float_array (matplotlib\image.py)
  8.00%   8.00%   0.080s    0.080s  bincount (<__array_function__ internals>)
  0.00%   0.00%   0.080s    0.080s  _sum (numpy\core\_methods.py)
  7.00%  30.00%   0.070s    0.380s  remove_small_objects (skimage\morphology\misc.py)
  0.00%   0.00%   0.060s    0.070s  _path_join (<frozen importlib._bootstrap_external>)
  0.00%   0.00%   0.060s    1.06s   forward (torch\nn\modules\container.py)
  0.00%   0.00%   0.060s    1.34s   _call_impl (torch\nn\modules\module.py)
  0.00%   0.00%   0.060s    1.34s   forward (model.py)
  0.00%   0.00%   0.060s    0.060s  relu (torch\nn\functional.py)
  0.00%   0.00%   0.040s    0.040s  _create_fn (dataclasses.py)
  0.00%  10.00%   0.040s    0.220s  remove_small_holes (skimage\morphology\misc.py)
  0.00%   0.00%   0.040s    0.120s  get_tiles (PngTools.py)
  0.00%   0.00%   0.030s    1.34s   _wrapped_call_impl (torch\nn\modules\module.py)
  0.00%   0.00%   0.030s    0.030s  lru_cache (functools.py)

Press Control-C to quit, or ? for help.

process 14340 ended
```

Рисунок 4.2 – Результати аналізу Py-Spy

%Own та OwnTime вказують на час, витрачений тільки певною функцією, без урахування викликів інших функцій/підфункцій.

%Total та TotalTime включають час, витрачений як на саму функцію, так і на всі виклики підфункцій.

З отриманих результатів можна дійти висновку, що програмне забезпечення працює досить ефективно, на що також вказує параметр Active. Він відображає відсоток часу, коли програма була активна (не перебувала в стані простою). У даному випадку, значення становить 100% часу.

4.2 Опис процесів тестування

Тестування виконується згідно документу «Програма та методика тестування».

Було виконане мануальне тестування програмного забезпечення, опис відповідних тестів наведено у таблицях 4.1 – 4.15.

Таблиця 4.1 – Тест 1.1 Вибір зображення для обробки

Початковий стан системи	Користувач відкрив термінал для використання CLI
Вхідні дані	ID зображення
Опис проведення тесту	Вводиться основна команда запуску (“python main.py”), після чого задається параметр “imID” типу String, що представляє собою ID необхідного зображення для обробки. Далі натискається кнопка “Enter” на клавіатурі для запуску.
Очікуваний результат	Якщо зображення не завантажено локально, тоді починається процес його завантаження з серверу PDS. Якщо зображення вже існує, тоді виводиться відповідне повідомлення й відбувається подальший процес обробки.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.2 – Тест 2.1 Вибір директорії з даними

Початковий стан системи	Користувач відкрив термінал для використання CLI
Вхідні дані	ID зображення, шлях до директорії з даними

Продовження таблиці 4.2

Опис проведення тесту	Вводиться основна команда запуску (“python main.py”), задається параметр “imID” для вибору необхідного зображення, після чого задається параметр “STORAGE_PATH” типу String, що представляє собою шлях до бажаної директорії сховища даних. Далі натискається кнопка “Enter” на клавіатурі для запуску.
Очікуваний результат	Якщо вказана директорія вже існує, тоді одразу буде виконуватись обробка вказаного зображення. Якщо вказаної директорії не існує, тоді вона буде створена, включаючи всі піддиректорії, після чого розпочнеться процес обробки зображення.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.3 – Тест 3.1 Вибір конфігураційного файлу

Початковий стан системи	Користувач відкрив термінал для використання CLI
Вхідні дані	ID зображення, шлях до конфігураційного файлу
Опис проведення тесту	Вводиться основна команда запуску (“python main.py”), задається параметр “imID” для вибору необхідного зображення, після чого задається параметр “CONFIG_PATH” типу String, що представляє собою шлях до конфігураційного файлу. Далі натискається кнопка “Enter” на клавіатурі для запуску.
Очікуваний результат	Якщо вказаний конфігураційний файл існує, тоді одразу буде виконуватись обробка вказаного зображення. Якщо шлях до конфігураційного файлу вказано невірно, або файлу не існує, тоді буде виведено повідомлення про помилку.

Продовження таблиці 4.3

Фактичний результат	Збігається з очікуваним.

Таблиця 4.4 – Тест 4.1 Вибір натренованої моделі UNet

Початковий стан системи	Користувач відкрив термінал для використання CLI
Вхідні дані	ID зображення, шлях до моделі UNet
Опис проведення тесту	Вводиться основна команда запуску (“python main.py”), задається параметр “imID” для вибору необхідного зображення, після чого задається параметр “MODEL_PATH” типу String, що представляє собою шлях до необхідної моделі нейромережі UNet. Далі натискається кнопка “Enter” на клавіатурі для запуску.
Очікуваний результат	Якщо вказана модель UNet існує, тоді одразу буде виконуватись обробка вказаного зображення. Якщо шлях до моделі UNet вказано невірно, або файлу моделі не існує, тоді буде виведено повідомлення про помилку.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.5 – Тест 5.1 Вибір певного каналу у зображенні

Початковий стан системи	Користувач відкрив термінал для використання CLI
Вхідні дані	ID зображення, номер каналу у зображенні

Продовження таблиці 4.5

Опис проведення тесту	Вводиться основна команда запуску (“python main.py”), задається параметр “imID” для вибору необхідного зображення, після чого задається параметр “BAND” типу Integer, що представляє собою номер каналу, який буде зчитано з растрового зображення. Далі натискається кнопка “Enter” на клавіатурі для запуску.
Очікуваний результат	Якщо вказаний номер каналу є доступним у зображенні, тоді його буде використано в подальшій обробці. Якщо вказаний номер каналу відсутній у зображенні, тоді буде виведено відповідне повідомлення про помилку.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.6 – Тест 6.1 Вибір коефіцієнту стискання зображення

Початковий стан системи	Користувач відкрив термінал для використання CLI
Вхідні дані	ID зображення, значення коефіцієнту стискання
Опис проведення тесту	Вводиться основна команда запуску (“python main.py”), задається параметр “imID” для вибору необхідного зображення, після чого задається параметр “RESIZE_FACTOR” типу Float, що представляє собою коефіцієнт зменшення розміру (стиснення) зображення у форматі JP2. Далі натискається кнопка “Enter” на клавіатурі для запуску.
Очікуваний результат	Растрове зображення JP2 буде зменшене (стиснуте) в необхідну кількість разів, відповідно до вказаного значення параметра.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.7 – Тест 7.1 Вибір розміру плиток, на які розподіляється зображення

Початковий стан системи	Користувач відкрив термінал для використання CLI
Вхідні дані	ID зображення, значення розміру плиток
Опис проведення тесту	Вводиться основна команда запуску (“python main.py”), задається параметр “imID” для вибору необхідного зображення, після чого задається параметр “TILE_SIZE” типу Integer, що представляє собою розмір плиток, на які розподіляється стиснуте зображення. Далі натискається кнопка “Enter” на клавіатурі для запуску.
Очікуваний результат	Якщо вказане значення розміру плитки не кратне 2, тоді буде виведено повідомлення про помилку. Якщо значення кратне 2, тоді процес обробки пройде успішно.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.8 – Тест 8.1 Відображення метаданих

Початковий стан системи	Користувач відкрив термінал для використання CLI
Вхідні дані	ID зображення, прапорець активації відображення метаданих
Опис проведення тесту	Вводиться основна команда запуску (“python main.py”), задається параметр “imID” для вибору необхідного зображення, після чого задається параметр “STATS” який слугує прапорцем командної строки, що сигналізує про бажання користувача увімкнути виведення статистичних метаданих для обраного растрового зображення. Далі натискається кнопка “Enter” на клавіатурі для запуску.

Продовження таблиці 4.8

Очікуваний результат	Якщо даний параметр вказано при запуску, тоді здійснюється виведення метаданих для обраного растрового зображення. Якщо даний параметр не вказано, тоді виведення метаданих не відбувається.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.9 – Тест 9.1 Вказання загальної кількості каналів у зображенні

Початковий стан системи	Користувач відкрив термінал для використання CLI
Вхідні дані	ID зображення, значення загальної кількості каналів у зображенні
Опис проведення тесту	Вводиться основна команда запуску (“python main.py”), задається параметр “imID” для вибору необхідного зображення, після чого задається параметр “NUM_CHANNELS” типу Integer, що представляє собою значення загальної кількості каналів у вхідному растровому зображенні. Далі натискається кнопка “Enter” на клавіатурі для запуску.
Очікуваний результат	Якщо значення кількості каналів у вхідному зображенні вказано вірно, тоді процес обробки пройде успішно. Якщо вказане значення кількості каналів не відповідає їх фактичній кількості у растровому зображенні, тоді буде виведене відповідне повідомлення про помилку.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.10 – Тест 10.1 Вказання кількості семантичних класів

Початковий стан системи	Користувач відкрив термінал для використання CLI
Вхідні дані	ID зображення, значення загальної кількості каналів у зображенні
Опис проведення тесту	Вводиться основна команда запуску (“python main.py”), задається параметр “imID” для вибору необхідного зображення, після чого задається параметр “NUM_CLASSES” типу Integer, що представляє собою значення кількості семантичних класів у згенерованій UNet бінарній масці. Далі натискається кнопка “Enter” на клавіатурі для запуску.
Очікуваний результат	Якщо значення кількості семантичних класів, які необхідно виявити у зображенні вказано вірно, тоді процес обробки пройде успішно. Якщо вказане значення кількості семантичних класів не відповідає фактичному, тоді буде виведене відповідне повідомлення про помилку.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.11 – Тест 11.1 Встановлення порогу чутливості

Початковий стан системи	Користувач відкрив термінал для використання CLI
Вхідні дані	ID зображення, значення порогу чутливості

Продовження таблиці 4.11

Опис проведення тесту	Вводиться основна команда запуску (“python main.py”), задається параметр “imID” для вибору необхідного зображення, після чого задається параметр “THRESHOLD” типу Float, що представляє собою значення порогу чутливості для конвертації матриці ймовірностей у бінарну маску. Далі натискається кнопка “Enter” на клавіатурі для запуску.
Очікуваний результат	Якщо значення порогу вказано вірно, в діапазоні від 0 до 1 не включно, тоді процес обробки пройде успішно. Якщо вказане значення дорівнює 1, або більше, тоді буде виведено відповідне повідомлення про помилку.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.12 – Тест 12.1 Встановлення значення для заповнення прогалин

Початковий стан системи	Користувач відкрив термінал для використання CLI
Вхідні дані	ID зображення, значення для заповнення прогалин
Опис проведення тесту	Вводиться основна команда запуску (“python main.py”), задається параметр “imID” для вибору необхідного зображення, після чого задається параметр “GAP_SIZE” типу Integer, що представляє собою значення, менше якого усі прогалини у бінарній масці будуть заповнені пікселями. Далі натискається кнопка “Enter” на клавіатурі для запуску.

Продовження таблиці 4.12

Очікуваний результат	Збільшення або зменшення значення даного параметру, напряму впливає на кінцеві результати обробки. Можна спостерігати візуальні зміни з отриманого, в результаті обробки, зображення з визначеними контурами.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.13 – Тест 13.1 Встановлення значення для видалення острівців

Початковий стан системи	Користувач відкрив термінал для використання CLI
Вхідні дані	ID зображення, значення для видалення острівців
Опис проведення тесту	Вводиться основна команда запуску (“python main.py”), задається параметр “imID” для вибору необхідного зображення, після чого задається параметр “ISLAND_SIZE” типу Integer, що представляє собою значення, менше якого усі об’єкти (острівці) у бінарній масці будуть видалені. Далі натискається кнопка “Enter” на клавіатурі для запуску.
Очікуваний результат	Збільшення або зменшення значення даного параметру, напряму впливає на кінцеві результати обробки. Можна спостерігати візуальні зміни з отриманого, в результаті обробки, зображення з визначеними контурами.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.14 – Тест 14.1 Встановлення відстані між точками, які формують контур

Початковий стан системи	Користувач відкрив термінал для використання CLI

Продовження таблиці 4.14

Вхідні дані	ID зображення, значення відстані між точками, які формують контур
Опис проведення тесту	Вводиться основна команда запуску (“python main.py”), задається параметр “imID” для вибору необхідного зображення, після чого задається параметр “CNT_DENSITY” типу Integer, що представляє собою значення відстані між точками, які формують контур. Далі натискається кнопка “Enter” на клавіатурі для запуску.
Очікуваний результат	Збільшення або зменшення значення даного параметру, напряму впливає на кінцеві результати обробки. Можна спостерігати візуальні зміни з отриманого, в результаті обробки, зображення з визначеними контурами.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.15 – Тест 15.1 Перегляд результатів обробки

Початковий стан системи	Користувач відкрив термінал для використання CLI
Вхідні дані	ID зображення, інші доступні параметри
Опис проведення тесту	Вводиться основна команда запуску (“python main.py”), задається параметр “imID” для вибору необхідного зображення, після чого задаються інші доступні параметри, або не задаються. Далі натискається кнопка “Enter” на клавіатурі для запуску.

Продовження таблиці 4.15

Очікуваний результат	Обробка зображення проходить успішно, після чого у відповідних директоріях сховища з’являються NPY файл, що зберігає список координат точок визначеного контуру та PNG файл, що представляє собою зображення з графічно визначеним контуром відповідного рельєфу. Ці файли доступні користувачу для перегляду та подальшого використання.
Фактичний результат	Збігається з очікуваним.

4.3 Опис контрольного прикладу

В якості контрольного прикладу обрано процес обробки зображення із виведенням метаданих та використанням параметру для встановлення відстані між точками, які формують контур.

Користувач, використовуючи CLI, вводить основну команду запуску (рисунок 4.3).

```
(contours) PS C:\Users\Nikita\Desktop\diploma\ReliefContours> python main.py
```

Рисунок 4.3 – Введення команди запуску

Користувач, використовуючи CLI, вказує ID необхідного зображення для обробки (рисунок 4.4).

```
(contours) PS C:\Users\Nikita\Desktop\diploma\ReliefContours> python main.py --imID ESP_018244_2655
```

Рисунок 4.4 – Вказання ID зображення на обробку

Користувач, використовуючи CLI, встановлює прапорець для виведення метаданих для обраного ним зображення (рисунок 4.5).

```
(contours) PS C:\Users\Nikita\Desktop\diploma\ReliefContours> python main.py --imID ESP_018244_2655 --STATS
```

Рисунок 4.5 – Встановлення прапорцю для виведення метаданих

Користувач, використовуючи CLI, встановлює параметр значення відстані між точками, які формують контур (рисунок 4.6).

```
(contours) PS C:\Users\Nikita\Desktop\diploma\ReliefContours> python main.py --imID ESP_018244_2655 --STATS --CNT_DENSITY 200
```

Рисунок 4.6 – Встановлення значення відстані між точками

Після вказання необхідних параметрів, користувач натискає на клавіатурі кнопку “Enter” для запуску процесу обробки зображення.

Якщо вказане зображення не завантажено локально, тоді розпочнеться процес його завантаження (рисунок 4.7)

```
Downloading :: ESP_018244_2655
```

Рисунок 4.7 – Повідомлення про завантаження зображення

Якщо обране зображення вже було попередньо завантажене, тоді виведеться повідомлення про його наявність (рисунок 4.8).

```
../data/images/JP2/ESP_018244_2655_RED.JP2 already exists
```

Рисунок 4.8 – Повідомлення про наявність зображення

Далі відбувається вивід метаданих для обраного користувачем зображення, оскільки ним було встановлено відповідний прапорець (рисунок 4.9).

```
Driver: JP2OpenJPEG/JPEG-2000 driver based on OpenJPEG library
Size is 49724 x 49773 x 1
Projection is PROJCS["Polar_Stereographic MARS",GEOGCS["GCS_MARS",DATUM["unnamed",SPHEROID["unnamed",3376200,0]
],PRIMEM["Reference meridian",0],UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","9122"]]],PROJECTION["Polar_
Stereographic"],PARAMETER["latitude_of_origin",90],PARAMETER["central_meridian",0],PARAMETER["scale_factor",1],
PARAMETER["false_easting",0],PARAMETER["false_northing",0],UNIT["metre",1],AXIS["Easting",SOUTH],AXIS["Northing
",SOUTH]]
Origin = (-51133.125, 277682.125)
Pixel Size = (0.25, -0.25)
Band Type=UInt16
Min=0, Max=1023
```

Рисунок 4.9 – Виведення метаданих для оброблюваного зображення

Потім виводиться повідомлення про успішне завантаження моделі нейромережі UNet (рисунок 4.10)

```
Model loaded successfully
```

Рисунок 4.10 – Повідомлення про завантаження моделі UNet

І наостанок виводиться значення часу, за яке було виконано процес обробки зображення (рисунок 4.11).

```
Processing Time: 9.3 seconds
```

Рисунок 4.11 – Виведення часу обробки зображення

Після успішного завершення процесу обробки, користувач отримує результати у вигляді NPY файлу, що зберігає список координат точок визначеного контуру та PNG файлу, що представляє собою зображення з графічно визначеним контуром відповідного рельєфу. Ці файли зберігаються у відповідних директоріях (рисунок 4.12).

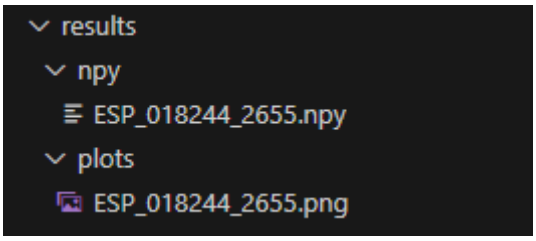


Рисунок 4.12 – Збережені файли результатів обробки

Користувач може переглянути результат у вигляді масиву координат точок контуру, що збережені в NPY файлі (рисунок 4.13).

```
import numpy as np
data = np.load('ESP_018244_2655.npy')

# для прикладу виведено 3 перші і 3 останні
np.set_printoptions(threshold=6)
print (data)
```

✓ 0.0s

```
[[4420 2558]
 [4418 2670]
 [4526 2743]
 ...
 [4681 2812]
 [4546 2671]
 [4432 2564]]
```

Рисунок 4.13 – Отриманий масив координат точок контуру

Користувач також має змогу переглянути отриманий результат у вигляді PNG зображення з графічно визначеним контуром відповідного рельєфу (рисунок 4.14).

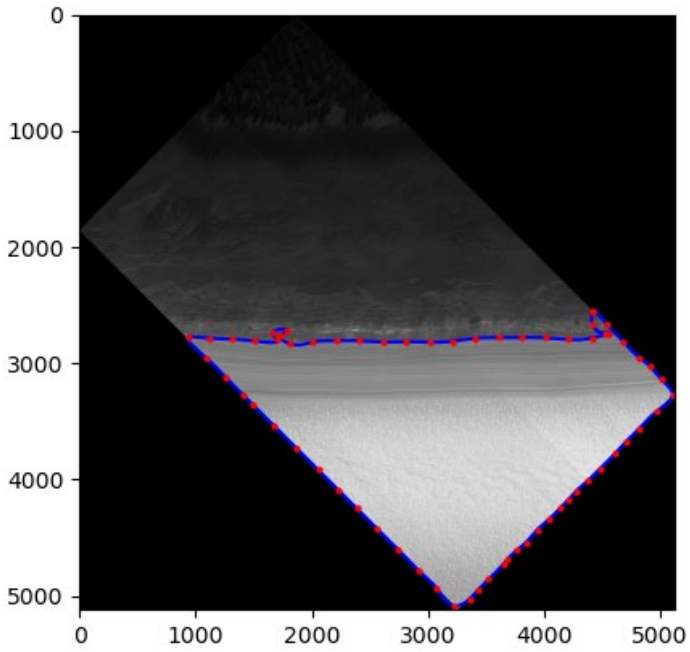


Рисунок 4.14 – Зображення з графічно визначеним контуром

Отримані результати готові для подальшого використання в інтересах користувача.

Висновки до розділу

В цьому розділі було проведено аналіз якості та тестування розробленого програмного забезпечення.

У підрозділі «Аналіз якості ПЗ» було описано мету тестувань та обрані метрики для оцінки якості ПЗ. Було використано статичний аналізатор SonarCloud та динамічний аналізатор Py-Spy. За результатами проведеного ними аналізу, було зроблено висновки щодо високого рівня якості розробленого програмного забезпечення.

У підрозділі «Опис процесів тестування» в таблицях було описано мануальне тестування, що було проведене відповідно до функціональних вимог. Було зазначено, що всі фактичні результати збіглись із очікуваними.

У підрозділі «Опис контрольного прикладу» було покроково описано та проілюстровано процес обробки зображення із відповідними вихідними результатами.

5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

Розроблена бібліотека має представляти собою програмне забезпечення з відкритим кодом, доступне для використання усіма заохоченими користувачами. З цієї причини, платформою для розгортання було обрано GitHub [23]. Дана платформа підтримує відкритість, що дозволяє будь-кому переглядати, використовувати, і навіть покращувати код. GitHub також надає потужні інструменти для керування версіями з використанням Git [24], що полегшує відстеження змін та повернення до попередніх версій.

Процес розгортання було виконано за наступними етапами:

а) створення репозиторію на платформі GitHub (рисунок 5.1);

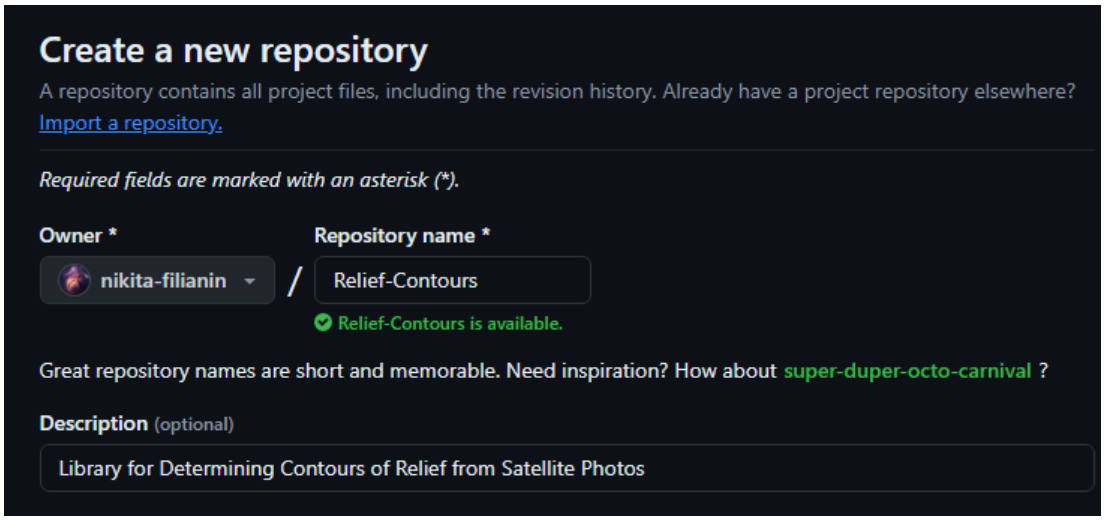


Рисунок 5.1 – Створення репозиторію GitHub

б) ініціалізація Git всередині локального репозиторію (рисунок 5.2);

```
(contours) PS C:\Users\Nikita\Desktop\diploma\ReliefContours> git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in C:/Users/Nikita/Desktop/diploma/ReliefContours/.git/
(contours) PS C:\Users\Nikita\Desktop\diploma\ReliefContours>
```

Рисунок 5.2 – Ініціалізація локального Git репозиторію

в) перейменування поточної гілки розробки на “main” (рисунок 5.3);

```
(contours) PS C:\Users\Nikita\Desktop\diploma\ReliefContours> git branch -M main
```

Рисунок 5.3 – Перейменування гілки на “main”

г) перегляд поточного стану робочої директорії для подальшої індексації файлів (рисунок 5.4);

```
(contours) PS C:\Users\Nikita\Desktop\diploma\ReliefContours> git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        Configs/
        Core/
        EnvConfigs/
        README.md
        UNet/
        main.ipynb
        main.py

nothing added to commit but untracked files present (use "git add" to track)
```

Рисунок 5.4 – Перегляд поточного стану робочої директорії

д) індексація всіх необхідних файлів проєкту (рисунок 5.5);

```
(contours) PS C:\Users\Nikita\Desktop\diploma\ReliefContours> git add .
```

Рисунок 5.5 – Індексація всіх необхідних файлів

е) перегляд поточного стану робочої директорії для подальшої фіксації всіх змін в індексованих файлах (рисунок 5.6);

```
(contours) PS C:\Users\Nikita\Desktop\diploma\ReliefContours> git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   .gitignore
        new file:   Configs/config.yaml
        new file:   Configs/config_parser.py
        new file:   Core/JP2Tools.py
        new file:   Core/PngTools.py
        new file:   Core/utility.py
        new file:   EnvConfigs/contours.yml
        new file:   README.md
        new file:   UNet/model.py
        new file:   UNet/weights/UNet_model.pth
        new file:   main.ipynb
        new file:   main.py
```

Рисунок 5.6 – Перегляд стану директорії після індексації

ж) фіксація (“commit”) змін в локальному репозиторії (рисунок 5.7);

```
(contours) PS C:\Users\Nikita\Desktop\diploma\ReliefContours> git commit -m "initial commit"
[main (root-commit) 499ebff] initial commit
 12 files changed, 886 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 Configs/config.yaml
 create mode 100644 Configs/config_parser.py
 create mode 100644 Core/JP2Tools.py
 create mode 100644 Core/PngTools.py
 create mode 100644 Core/utility.py
 create mode 100644 EnvConfigs/contours.yml
 create mode 100644 README.md
 create mode 100644 UNet/model.py
 create mode 100644 UNet/weights/UNet_model.pth
 create mode 100644 main.ipynb
 create mode 100644 main.py
```

Рисунок 5.7 – Фіксація змін в локальному репозиторії

з) вказання віддаленого репозиторію GitHub для встановлення зв’язку з локальним (рисунок 5.8);

```
git remote add origin https://github.com/nikita-filianin/Relief-Contours.git
```

Рисунок 5.8 – Вказання віддаленого репозиторію GitHub

и) завантаження зафіксованих змін (даних) з локальної гілки у віддалений репозиторій GitHub (рисунок 5.9).

```
(contours) PS C:\Users\Nikita\Desktop\diploma\ReliefContours> git push -u origin main
Enumerating objects: 19, done.
Counting objects: 100% (19/19), done.
Delta compression using up to 12 threads
Compressing objects: 100% (16/16), done.
Writing objects: 100% (19/19), 20.58 MiB | 1.68 MiB/s, done.
Total 19 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/nikita-filianin/ReIief-Contours.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

Рисунок 5.9 – Завантаження у віддалений репозиторій GitHub

На виході отримано репозиторій на платформі GitHub, що містить усі необхідні файли бібліотеки (рисунок 5.10).

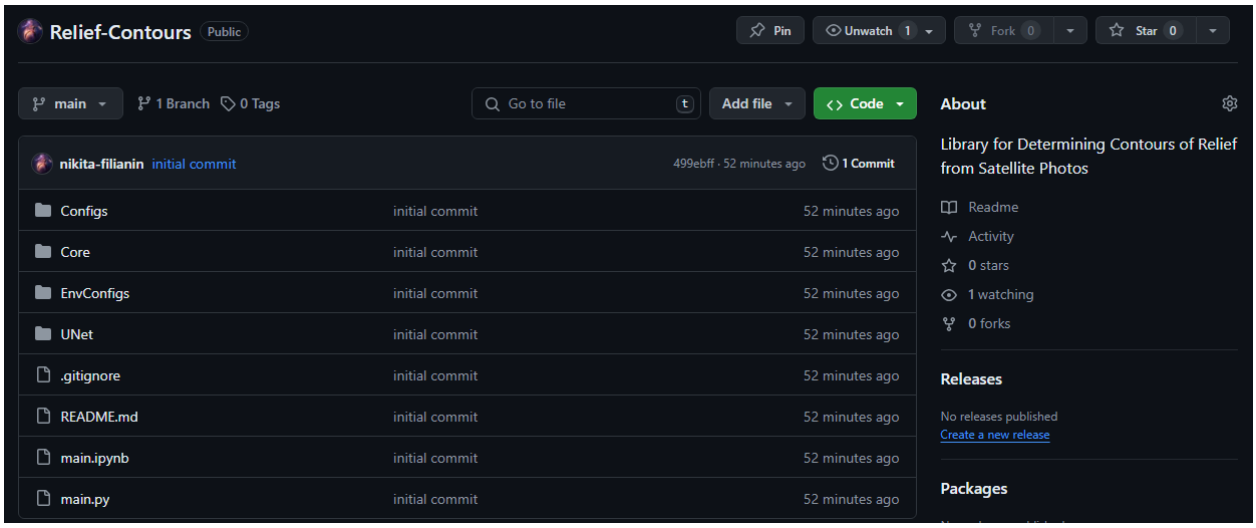


Рисунок 5.10 – Репозиторій GitHub з усіма файлами бібліотеки

5.2 Супровід програмного забезпечення

Інструкція користувача наведена в окремому документі.

Користувачі будуть мати змогу отримати актуальну версію програмного забезпечення з кожними новими змінами, що були завантажені в гілку “main”. Якщо користувач клонував собі репозиторій командою “git clone”, тоді він завжди зможе отримати останню версію програмного забезпечення скориставшись командою “git pull”. Дана команда використовується для оновлення локальної копії репозиторію шляхом отримання останніх змін з віддаленого репозиторію і об’єднання їх з поточною локальною гілкою.

Для гарантії забезпечення користувачів саме стабільно працюючими версіями бібліотеки використовується розділ релізів (“Releases”) на сторінці репозиторію GitHub (рисунок 5.11).

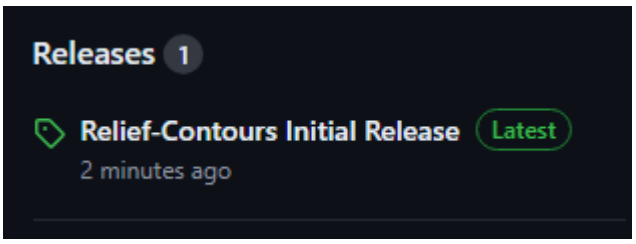


Рисунок 5.11 – Розділ релізів

Реліз версії гарантують для користувачів:

– надійність: стабільні і перевірені версії програмного забезпечення;

– прозорість: ознайомлення з усіма змінами в нових версіях;

– зручність: доступ до необхідних файлів, зібраних в одному місці.

Приклад релізу представлено на рисунку 5.12.

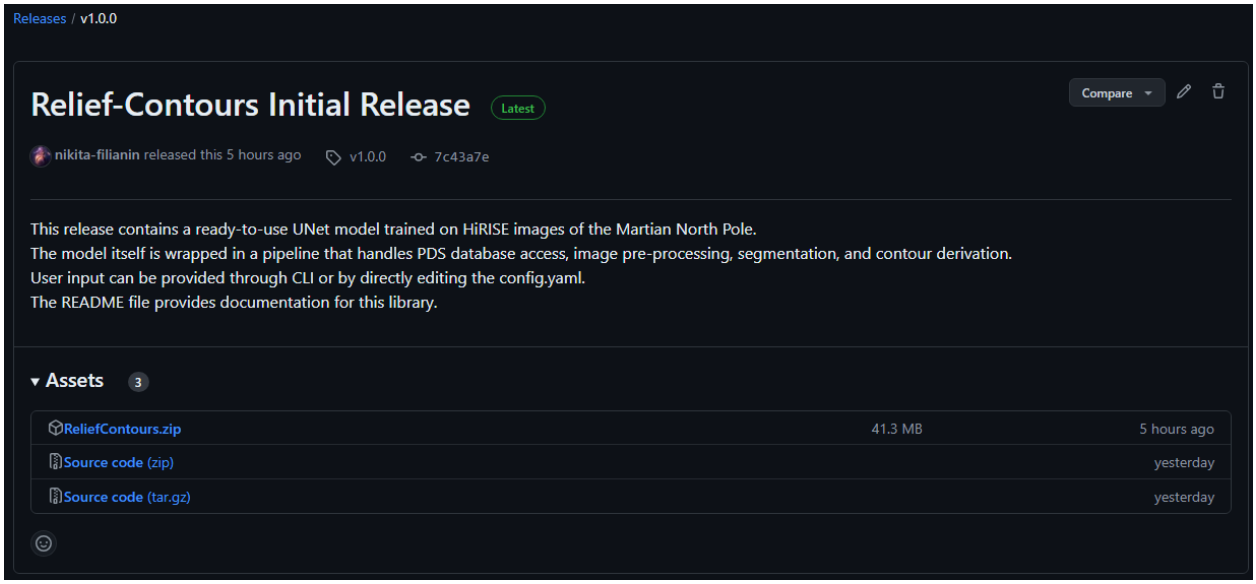


Рисунок 5.12 – Реліз-версія бібліотеки

Висновки до розділу

В цьому розділі було описано аспекти розгортання та супроводу програмного забезпечення.

У підрозділі «Розгортання програмного забезпечення» було описано вибір платформи та подальший поетапний процес розгортання на ній програмного забезпечення. Виконання кожного з етапів підкріплено відповідними ілюстраціями.

У підрозділі «Супровід програмного забезпечення» було описано варіанти отримання користувачами останніх актуальних версій бібліотеки.

ВИСНОВКИ

В результаті виконання дипломного проєкту було спроєктовано та розроблено бібліотеку для визначення контурів рельєфу з супутникових фотографій. Основною задачею розробленого продукту є виявлення контурів геологічних структур (семантичних класів) з растрових зображень та отримання на виході координат точок, які відповідно формують ці самі контури.

Для реалізації програмного забезпечення було використано мову програмування Python, зокрема через чисельність підтримуваних нею наукових бібліотек. В якості середовища розробки було обрано Visual Studio Code. Розроблене рішення являє собою комплексний підхід. Він включає в себе статистичні методи обробки, такі як ті, що реалізовані у бібліотеках OpenCV та scikit-image, та використання нейромережі UNet для семантичної сегментації. Така комбінація дозволяє ефективно використовувати переваги обох методів: швидкість та простоту першого для попередньої обробки даних і точність та автономність другого для визначення точних контурів рельєфу.

Після реалізації програмного продукту, його було протестовано на відповідність до висунутих функціональних вимог та інших поставлених задач. Таким чином було зроблено висновки щодо правильності його роботи.

Готовий вихідний код та інші файли бібліотеки було розгорнуто у загальному доступі на платформі GitHub. Таким чином, дана робота представляє собою програмне забезпечення із відкритим кодом, доступним для використання усіма заохоченими користувачами.

Оскільки дані з камери HiRISE вважаються відносно новими та використовуються у великій кількості поточних досліджень, дана робота має високу науково-технічну цінність та великий потенціал для подальших напрацювань. Розроблена бібліотека має значно полегшити роботу геодезистів і планетарних вчених, які досліджують клімат і географію Марса.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) HiRISE | Outreach. *HiRISE | High Resolution Imaging Science Experiment*. URL: https://www.uahirise.org/epo (дата звернення: 18.03.2024).

2) What is Computer Vision? | IBM. *IBM - United States*. URL: https://www.ibm.com/topics/computer-vision (дата звернення: 19.03.2024).

3) Python 3.12.4 Documentation. *3.12.4 Documentation*. URL: https://docs.python.org/3/ (дата звернення: 24.03.2024).

4) What is GDAL? – GDAL documentation. *GDAL – GDAL documentation*. URL: https://gdal.org/about.html (дата звернення: 27.03.2024).

5) OpenCV About. *OpenCV*. URL: https://opencv.org/about/ (дата звернення: 01.04.2024).

6) scikit-image: Image processing in Python – scikit-image. *scikit-image: Image processing in Python – scikit-image*. URL: https://scikit-image.org/ (дата звернення: 02.04.2024).

7) What is UNET? - Idiot Developer. *Idiot Developer*. URL: https://idiotdeveloper.com/what-is-unet/ (дата звернення: 07.04.2024).

8) Rosencrance L. What is a PNG (Portable Network Graphics)?. *WhatIs*. URL: https://www.techtarget.com/whatis/definition/PNG-Portable-Network-Graphics (дата звернення: 13.05.2024).

9) JP2 - Image File Format. *File Format Docs*. URL: https://docs.fileformat.com/image/jp2/ (дата звернення: 25.03.2024).

10) numpy.lib.format – NumPy v2.1.dev0 Manual. *NumPy -*. URL: https://numpy.org/devdocs/reference/generated/numpy.lib.format.html (дата звернення: 15.04.2024).

11) What is PyTorch?. *NVIDIA Data Science Glossary*. URL: https://www.nvidia.com/en-us/glossary/pytorch/ (дата звернення: 08.04.2024).

12) What is Conda? | Cloudsmith. *Cloudsmith*. URL: https://cloudsmith.com/blog/what-is-conda (дата звернення: 25.03.2024).

13) virtualenv. *virtualenv*. URL: https://virtualenv.pypa.io/en/latest/ (дата звернення: 24.03.2024).

14) Real Python. Pipenv: A Guide to the New Python Packaging Tool – Real Python. *Python Tutorials – Real Python*. URL: https://realpython.com/pipenv-guide/ (дата звернення: 24.03.2024).

15) Microsoft. Why Visual Studio Code?. *Visual Studio Code - Code Editing. Redefined*. URL: https://code.visualstudio.com/docs/editor/whyvscode (дата звернення: 19.03.2024).

16) Patel M. Understanding the Rectified Linear Unit (ReLU): A Key Activation Function in Neural Networks. *Medium*. URL: https://medium.com/@meetkp/understanding-the-rectified-linear-unit-relu-a-key-activation-function-in-neural-networks-28108fba8f07 (дата звернення: 07.04.2024).

17) skimage.morphology – skimage 0.23.2 documentation. *scikit-image: Image processing in Python – scikit-image*. URL: https://scikit-image.org/docs/stable/api/skimage.morphology.html#skimage.morphology.remove_small_holes (дата звернення: 18.04.2024).

18) skimage.morphology – skimage 0.23.2 documentation. *scikit-image: Image processing in Python – scikit-image*. URL: https://scikit-image.org/docs/stable/api/skimage.morphology.html#skimage.morphology.remove_small_objects (дата звернення: 18.04.2024).

19) OpenCV: Contours : Getting Started. *OpenCV documentation index*. URL: https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html (дата звернення: 21.04.2024).

20) Index of /PDS/RDR/ESP/. *Index of /PDS/*. URL: https://hirise-pds.lpl.arizona.edu/PDS/RDR/ESP/ (дата звернення: 24.03.2024).

21) What SonarCloud can do. *SonarQube 10.5*. URL: https://docs.sonarsource.com/sonarcloud/discovering-sonarcloud/what-sonarcloud-can-do/ (дата звернення: 10.05.2024).

22) py-spy. *PyPI*. URL: https://pypi.org/project/py-spy/0.1.3 (дата звернення: 10.05.2024).

23) GitHub: Let’s build from here. *GitHub*. URL: https://github.com/ (дата звернення: 25.05.2024).

24) Git - What is Git?. *Git*. URL: https://www.git-scm.com/book/en/v2/Getting-Started-What-is-Git? (дата звернення: 25.05.2024).

ДОДАТОК А ЗВІТ ПОДІБНОСТІ

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1016307881

Дата перевірки:
01.06.2024 23:35:19 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
02.06.2024 07:06:28 EEST

ID користувача:
76913

Назва документа: ІТ-01_Філянін_ПЗ

Кількість сторінок: 58 Кількість слів: 9172 Кількість символів: 74948 Розмір файлу: 1.07 MB ID файлу: 1016104276

6.42%
Схожість

Найбільша схожість: 3.19% з джерелом з Бібліотеки (ID файлу: 1016099047)

2.39% Джерела з Інтернету | 131 ..... Сторінка 60

6.09% Джерела з Бібліотеки | 291 ..... Сторінка 61

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи | 1

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

БІБЛІОТЕКА ДЛЯ ВИЗНАЧЕННЯ КОНТУРІВ РЕЛЬЄФУ З СУПУТНИКОВИХ ФОТОГРАФІЙ

Текст програми

КПІ.ІТ-0321.045490.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олександр СТЕЛЬМАХ

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Нікіта ФІЛЯНІН

Київ – 2024

Посилання на репозиторій з повним текстом програмного коду
https://github.com/nikita-filianin/Relief-Contours

Файл config_parser.py

```
import os
import yaml
import torch

class AttrDict(dict):
    def __init__(self, *args, **kwargs):
        super(AttrDict, self).__init__(*args, **kwargs)
        self.__dict__ = self

class Config():
    def __init__(self, yml_path='config.yaml'):
        self.conf = AttrDict()
        with open(yml_path) as file:
            yaml_cfg = yaml.load(file, Loader=yaml.FullLoader)
        self.conf.update(yaml_cfg)

    def getConfig(self, args: dict) -> dict:
        """
        Initializes some additional internal variables; based on user input in
config.yaml
        """
        args = {key: value for key, value in args.items() if value != None}
        self.conf.update(args)

        # SET PATHS
        self.conf.JP2_DIR = f'{self.conf.STORAGE_PATH}/images/JP2'
        self.conf.JP2_FPATH = f'{self.conf.JP2_DIR}/{self.conf.imID}_RED.JP2'

        self.conf.PNG_DIR = f'{self.conf.STORAGE_PATH}/images/png'
        self.conf.PNG_FPATH = f'{self.conf.PNG_DIR}/{self.conf.imID}_RED.png'

        self.conf.RESULTS_DIR =f'{self.conf.STORAGE_PATH}/results/npy'
        self.conf.RESULTS_FPATH = f'{self.conf.RESULTS_DIR}/{self.conf.imID}.npy'

        self.conf.PLOT_DIR = f'{self.conf.STORAGE_PATH}/results/plots'
        self.conf.PLOT_FPATH = f'{self.conf.PLOT_DIR}/{self.conf.imID}.png'

        self.conf.DEVICE = 'cuda:0' if torch.cuda.is_available() else 'cpu'

        return self.conf
```

Файл config.yaml

```
---
# Paths
STORAGE_PATH: '../data'                                # Path to the data folder
MODEL_PATH: 'UNet/weights/UNet_model.pth'              # Path to the saved UNet
weights (.pth file)

# Gdal
BAND: 1                                                # Band to read from the raster
file
RESIZE_FACTOR: 0.1                                     # Downscaling factor applied to
the JP2 image
TILE_SIZE: 512                                         # Size of the tiles cut from
donwsacled .png (pixels)

# Unet parameters
NUM_CHANNELS: 1                                        # Number of channels in the
input image
NUM_CLASSES: 1                                         # Number of classes in the
output mask
THRESHOLD: 0.5                                         # Threshold for the output
mask; eveything below this value is set to 0, everything above to 1

# skimage.morpohlogy & edge detection
GAP_SIZE: 50000                                        # Any gap smaller than this
value will be filled (pixels)
ISLAND_SIZE: 50000                                     # Any island smaller than this
value will be removed (pixels)
CNT_DENSITY: 100                                       # Gap between knot points along
the contour (pixels); the higher the value, the less points are kept, the
smoother the contour.
```

Файл JP2Tools.py

```
import os
import numpy as np
from osgeo import gdal
gdal.UseExceptions()

class GdalUtils():
    def __init__(self, conf: dict) -> None:
        self.conf = conf
        self.rf = conf.RESIZE_FACTOR
        self.tsize = conf.TILE_SIZE
        self.bnum = conf.BAND
        self.pngpath = conf.PNG_FPATH

        self.dataset = gdal.Open(conf.JP2_FPATH, gdal.GA_ReadOnly)
```

```
        self.geotransform = self.dataset.GetGeoTransform()
        self.band = self.dataset.GetRasterBand(1)

        self.min_val, self.max_val = self.band.ComputeRasterMinMax(True)
        self.min_val, self.max_val = int(self.min_val), int(self.max_val)

        self.XSize, self.YSize, self.band_num = self.dataset.RasterXSize,
self.dataset.RasterYSize, self.dataset.RasterCount

    def get_stats(self):
        """
        Prints basic information about the image
        """
        print("Driver: {}/{}".format(self.dataset.GetDriver().ShortName,
                                     self.dataset.GetDriver().LongName))
        print("Size is {} x {} x {}".format(self.XSize,
                                            self.YSize,
                                            self.band_num))
        print("Projection is {}".format(self.dataset.GetProjection()))

        print("Origin = ({}, {})".format(self.geotransform[0],
self.geotransform[3]))
        print("Pixel Size = ({}, {})".format(self.geotransform[1],
self.geotransform[5]))

        print("Band Type={}".format(gdal.GetDataTypeName(self.band.DataType)))
        print("Min={}, Max={}".format(self.min_val, self.max_val))

    def downscale(self, save_format: str = 'png', rm_xml: bool = True) -> None:
        """
        Downscales the .JP2 image and saves it as .png
        Args:
            save_format (str): format of the saved image
            rm_xml (bool): whether to remove .aux.xml file
        Returns:
            None: saves the image as .png
        """
        if not os.path.exists(self.pngpath):
            W = np.round((self.XSize*self.rf)/self.tsize).astype(int)*self.tsize
            H = np.round((self.YSize*self.rf)/self.tsize).astype(int)*self.tsize
            gdal.Translate(self.pngpath, self.dataset, bandList = [self.bnum],
width = W, height = H, \
                           format = save_format, scaleParams = [[self.min_val,
self.max_val, self.min_val, 255]], outputType=gdal.GDT_Byte)

            if rm_xml:
                os.remove(f'{self.pngpath}.aux.xml')
        else:
            print(f'{self.pngpath} already exists')
```

Файл PngTools.py

```
import torch
import cv2 as cv
import numpy as np
from skimage import morphology
import matplotlib.pyplot as plt

from UNet.model import UNet

class PngUtils():
    def __init__(self, conf: dict) -> None:
        self.conf = conf
        self.tsize = conf.TILE_SIZE
        self.thalf = self.tsize//2
        self.tquat = self.tsize//4


        self.device = conf.DEVICE
        self.mpath = conf.MODEL_PATH
        self.threshold = conf.THRESHOLD
        self.gsize = conf.GAP_SIZE
        self.isize = conf.ISLAND_SIZE
        self.imID = conf.imID
        self.image = plt.imread(conf.PNG_FPATH)

    def get_tiles(self) -> tuple[np.ndarray, np.ndarray]:
        """
        Cuts .png image into overlaping 512x512 tiles
        Returns:
            np.array: array of tile corner (top left) coordinates (x, x+dx, y,
y+dy)
            np.array: array of tiles
        """
        mask = (self.image > 0)

        y1_arr = np.arange(0, self.image.shape[0] - self.thalf,
self.thalf).reshape(-1, 1)
        x1_arr = np.arange(0, self.image.shape[1] - self.thalf,
self.thalf).reshape(-1, 1)

        grid = np.array(np.meshgrid(x1_arr, y1_arr)).T.reshape(-1, 2)
        grid = np.insert(grid, 1, grid[:,0] + self.tsize, axis = 1)
        grid = np.insert(grid, 3, grid[:,2] + self.tsize, axis = 1)

        coord_lst = []
        tiles_lst = []
        for i in range(grid.shape[0]):
            x0, x1, y0, y1 = grid[i][0], grid[i][1], grid[i][2], grid[i][3]
            msk = mask[y0:y1, x0:x1]
```

```
            if msk.sum() > 0:
                coord_lst.append(grid[i])
                tiles_lst.append(self.image[y0:y1, x0:x1])

        return(np.array(coord_lst), np.array(tiles_lst))

    def generate_coordinate_arrs(self, coord_arr: np.ndarray) ->
tuple[np.ndarray, np.ndarray]:
        """
        Generates coordinates of the submask with respect to the original image
and the 512x512 tile it belongs to.
        Args:
            coord_arr (np.ndarray): Array of tile corner coordinates with respect
to the original image.
        Returns:
            img_coord_arr (np.ndarray): Array of submask corner coordinates (x,
x+dx, y, y+dy) with respect to the original image.
            tile_coord_arr (np.ndarray): Array of submask corner coordinates (x,
x+dx, y, y+dy) with respect to the tile it belongs to.
        """

        img_coord_arr = coord_arr.copy()
        tile_coord_arr = np.tile([0, self.tsize, 0, self.tsize],
(len(img_coord_arr), 1))
        for i in range(img_coord_arr.shape[1]):
            sign, val = (1, 0) if i%2 == 0 else (-1, img_coord_arr[:,i].max())

            img_coord_arr[:,i][img_coord_arr[:,i] != val] += sign*self.tquat
            tile_coord_arr[:,i][img_coord_arr[:,i] != val] += sign*self.tquat

        return (img_coord_arr, tile_coord_arr)


    def segment(self, coord_arr: np.ndarray, tile_arr: np.ndarray) -> np.ndarray:
        """
        Segments the image using the UNet model.
        Args:
            coord_arr (np.ndarray): Array of tile corner coordinates with respect
to the original image.
            tile_arr (np.ndarray): Array of 512x512 tiles.
        Returns:
            mask (np.ndarray): Binary segmentaion mask for the whole image.
        """

        model = UNet(self.conf).to(self.device)
        model.load_state_dict(torch.load(self.mpath,
map_location=torch.device(self.device))['state_dict'])
        model.eval()
        print('Model loaded successfully')

        mask = np.zeros_like(self.image)
```

```
        img_coord_arr, tile_coord_arr = self.generate_coordinate_arrs(coord_arr)

        with torch.no_grad():
            for img_coord, tile_coord, tile in zip(img_coord_arr, tile_coord_arr,
tile_arr):
                tile = torch.tensor(tile).float().to(self.device)
                tile = tile.unsqueeze(0).unsqueeze(0)

                predMask = model(tile).squeeze(0).squeeze(0)
                predMask = torch.relu(torch.sign(torch.sigmoid(predMask)-
self.threshold))
                predMask = predMask.cpu().numpy().astype(np.uint8)

                ix0, ix1, iy0, iy1 = img_coord
                tx0, tx1, ty0, ty1 = tile_coord
                mask[iy0:iy1, ix0:ix1] = predMask[ty0:ty1, tx0:tx1]

        mask = mask > 0
        mask = morphology.remove_small_holes(mask, area_threshold=self.gsize)
        mask = morphology.remove_small_objects(mask, min_size=self.isize)
        return mask

    def edge_detect(self, mask: np.ndarray, cnt_density: np.uint8) -> np.ndarray:
        """
        Detect contours of the largest object in the binary mask
        Args:
            mask (np.ndarray): Binary mask.
            point_density (int): Gap between knot points along the contour
(pixels);
                                 the higher the value, the less points are kept,
the smoother the contour.
        Returns:
            edge_points (np.ndarray): Array of edge points.
        """

        contours, _ = cv.findContours(mask.astype(np.uint8), cv.RETR_EXTERNAL,
cv.CHAIN_APPROX_NONE)

        if not contours:
            print('No contours found')
        else:
            edge_points = contours[0].reshape((-1, 2))
            idxs = np.arange(0, len(edge_points), cnt_density)
            edge_points = edge_points[idxs]

            return edge_points
```

Файл utility.py

```
import os
import urllib.request
import numpy as np

import matplotlib
import matplotlib.pyplot as plt
from scipy.interpolate import splprep, splev

def makeDirs(conf: dict) -> None:
    """
    Create directories for the project
    Args:
        conf (dict) - configuration dictionary
    """

    os.makedirs(conf.JP2_DIR, exist_ok=True)
    os.makedirs(conf.PNG_DIR, exist_ok=True)
    os.makedirs(conf.RESULTS_DIR, exist_ok=True)
    os.makedirs(conf.PLOT_DIR, exist_ok=True)

def downloadJP2(conf: dict) -> None:
    """
    Download the JP2 image from the PDS server
    Args:
        conf (dict): Configuration dictionary
    Return:
        None
    """

    if os.path.exists(conf.JP2_FPATH):
        print(f'{conf.JP2_FPATH} already exists')
    else:
        print(f'Downloading :: {conf.imID} \n')

        path = conf.imID.split('_')[0]
        ORB = conf.imID.split('_')[1][:4]

        URL = f'https://hirise-
pds.lpl.arizona.edu/PDS/RDR/{path}/ORB_{ORB}00_{ORB}99/{conf.imID}/{conf.imID}_RE
D.JP2'
        urllib.request.urlretrieve(URL, conf.JP2_FPATH)

def save_results(conf: dict, contours: np.ndarray) -> None:
    """
    Save the contour coordinates to a .npy file & Save the contour image
    Args:
        contours (np.ndarray): Array of contour coordinates
        conf (dict): Configuration dictionary
```

```
    """

    np.save(conf.RESULTS_FPATH, contours)

    image = plt.imread(conf.PNG_FPATH)
    tck, u = splprep(contours.T, u=None, s=0.0, per=1)
    u_new = np.linspace(u.min(), u.max(), 1000)
    x_new, y_new = splev(u_new, tck, der=0)

    plt.imshow(image, cmap='gray')
    plt.plot(x_new, y_new, 'b')
    plt.plot(contours[:, 0], contours[:, 1], 'o', color='red', markersize=2)
    plt.savefig(conf.PLOT_FPATH)
```

Файл model.py

```
import torch
import torch.nn as nn

class ConvBlock(nn.Module):
    def __init__(self, in_c, out_c):
        super().__init__()
        self.convblock = nn.Sequential(
            nn.Conv2d(in_c, out_c, kernel_size=3, stride = 1, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(out_c),
            nn.Conv2d(out_c, out_c, kernel_size=3, stride = 1, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(out_c)
        )

    def forward(self, inputs):
        x = self.convblock(inputs)
        return x


class Encoder(nn.Module):
    def __init__(self, in_c, out_c):
        super().__init__()
        self.conv = ConvBlock(in_c, out_c)
        self.pool = nn.MaxPool2d((2, 2))
        self.drop = nn.Dropout(0.1)

    def forward(self, inputs):
        skip = self.conv(inputs)
        x = self.drop(skip)
        x = self.pool(x)
        return x, skip


class Decoder(nn.Module):
```

```
    def __init__(self, in_c, out_c):
        super().__init__()
        self.up = nn.ConvTranspose2d(in_c, out_c, kernel_size=2, stride=2,
padding=0)
        self.conv = ConvBlock(out_c+out_c, out_c)
        self.drop = nn.Dropout(0.1)

    def forward(self, inputs, skip):
        x = self.up(inputs)
        x = torch.cat([x, skip], axis=1)
        x = self.conv(x)
        x = self.drop(x)
        return x


class UNet(nn.Module):
    def __init__(self, conf):
        super().__init__()
        self.cls = conf.NUM_CLASSES
        self.chl = conf.NUM_CHANNELS

        """ Encoder """
        self.enc1 = Encoder(self.chl, 16)
        self.enc2 = Encoder(16, 32)
        self.enc3 = Encoder(32, 64)
        self.enc4 = Encoder(64, 128)

        """ Bottleneck """
        self.bneck = ConvBlock(128, 256)

        """ Decoder """
        self.dec1 = Decoder(256, 128)
        self.dec2 = Decoder(128, 64)
        self.dec3 = Decoder(64, 32)
        self.dec4 = Decoder(32, 16)

        """ Segmenter"""
        self.out2 = nn.Conv2d(16, self.cls, kernel_size=1, padding=0)

    def forward(self, inputs):
        """ Encoder """
        x, skip1 = self.enc1(inputs)
        x, skip2 = self.enc2(x)
        x, skip3 = self.enc3(x)
        x, skip4 = self.enc4(x)

        """ Bottleneck """
        x = self.bneck(x)

        """ Decoder """
        x = self.dec1(x, skip4)
```

```
        x = self.dec2(x, skip3)
        x = self.dec3(x, skip2)
        x = self.dec4(x, skip1)

        """ Segmenter """
        x = self.out2(x)

        return x
```

Файл main.py

```
from Configs.config_parser import Config

from Core.JP2Tools import GdalUtils
from Core.PngTools import PngUtils
from Core.utility import makeDirs, downloadJP2, save_results

import argparse
import time


def main(conf: dict):
    makeDirs(conf)
    downloadJP2(conf)

    jp2_img = GdalUtils(conf)
    if conf.STATS:
        jp2_img.get_stats()
    jp2_img.downscale()

    png_img = PngUtils(conf)
    coord_arr, tile_arr = png_img.get_tiles()
    mask = png_img.segment(coord_arr, tile_arr)
    edge_points = png_img.edge_detect(mask, conf.CNT_DENSITY)
    save_results(conf, edge_points)

if __name__ == '__main__':

    parser = argparse.ArgumentParser(description='Available parameters to enter')

    # General args
    parser.add_argument('--imID', metavar = ':', type=str, required = True,
help='Image ID')
    parser.add_argument('--STORAGE_PATH', metavar = ':', type=str, help = 'Path
to data storage')
    parser.add_argument('--CONFIG_PATH', metavar = ':', default =
'Configs/config.yaml', type=str, help = 'Path to .yaml config file')
    parser.add_argument('--MODEL_PATH', metavar = ':', type=str, help = 'Path to
UNet .pth weights file')
```

```
    # Gdal args
    parser.add_argument('--BAND', metavar = ':', type=int, help='Band to read
from the raster file')
    parser.add_argument('--RESIZE_FACTOR', metavar = ':',
type=float,  help='Downscaling factor applied to the JP2 image')
    parser.add_argument('--TILE_SIZE', metavar = ':', type=int,  help='Size of
the tiles cut from downscaled .png (pixels)')
    parser.add_argument('--STATS', action='store_true', help='Show gdal
statistics?')

    # UNet args
    parser.add_argument('--NUM_CHANNELS', metavar = ':', type=int,  help='Number
of channels in the input image')
    parser.add_argument('--NUM_CLASSES', metavar = ':', type=int,  help='Number
of classes in the output mask')
    parser.add_argument('--THRESHOLD', metavar = ':', type=float, help='Threshold
for the output mask; eveything below this value is set to 0, everything above to
1')

    # skimage.morphology.morphology & edge detection args
    parser.add_argument('--GAP_SIZE', metavar = ':', type=int,  help='Any gap
smaller than this value will be filled (pixels)')
    parser.add_argument('--ISLAND_SIZE', metavar = ':', type=int,  help='Any
island smaller than this value will be removed (pixels)')
    parser.add_argument('--CNT_DENSITY', metavar = ':', type=int,  help='Gap
between knot points along the contour (pixels); the higher the value, the less
points are kept, the smoother the contour')

    args = (parser.parse_args()).__dict__

    conf = Config(args['CONFIG_PATH']).getConfig(args)

    start_time = time.time()

    main(conf)

    end_time = time.time()
    processing_time = end_time - start_time
    print(f'Processing Time: {processing_time:.1f} seconds')
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

БІБЛІОТЕКА ДЛЯ ВИЗНАЧЕННЯ КОНТУРІВ РЕЛЬЄФУ З СУПУТНИКОВИХ ФОТОГРАФІЙ

Програма та методика тестування

КПІ.ІТ-0321.045490.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олександр СТЕЛЬМАХ

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Нікіта ФІЛЯНІН

Київ – 2024

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є бібліотека для визначення контурів рельєфу з супутникових фотографій, що розроблена на мові програмування Python.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

– перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;

– перевірка збереження даних;

– знаходження проблем, помилок і недоліків з метою їх усунення;

– перевірка зручності інтерфейсу командного рядка.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

– статичне тестування – перевіряється програма разом з усією документацією, яка аналізується на предмет дотримання стандартів програмування;

– динамічне тестування – застосовується в процесі виконання програми. Коректність програмного засобу перевіряється на певній кількості тестів. При прогоні кожного з них збираються та аналізуються дані про проблеми та помилки в роботі програми;

– мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується мануально з використанням наскрізного тестування, з метою знаходження помилок та недоліків як у функціональній частині програмного забезпечення так і в зручності користування. Для того, щоб перевірити працездатність та відмовостійкість застосунку, необхідно провести наступні тестування:

- тестування на відповідність функціональним вимогам;
- тестування на виведення повідомлень про помилку, коли це необхідно;
- тестування на дотримання стандартів програмування;
- тестування збереження даних;
- тестування інтерфейсу командного рядка (CLI);
- тестування зручності використання.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

БІБЛІОТЕКА ДЛЯ ВИЗНАЧЕННЯ КОНТУРІВ РЕЛЬЄФУ З СУПУТНИКОВИХ ФОТОГРАФІЙ

Керівництво користувача

КПІ.ІТ-0321.045490.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олександр СТЕЛЬМАХ

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Нікіта ФІЛЯНІН

Київ – 2024

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

«Relief Contours» - це Python бібліотека для визначення контурів рельєфу з супутникових фотографій. Її основна задача – це виявлення контурів геологічних структур (семантичних класів) з растрових зображень та отримання на виході координат точок, які відповідно формують ці самі контури.

Бібліотека представлена у вигляді відкритого програмного коду, який розгорнуто на платформі GitHub.

Репозиторій містить готову до використання модель UNet, натреновану на зображеннях Північного полюса Марса, зроблених камерою HiRISE. Сама модель є елементом у розробленому pipeline, що включає отримання доступу до бази даних PDS для завантаження зображень, попередню обробку зображень, сегментацію та виведення контурів. Користувач може вводити дані через CLI, або шляхом безпосереднього редагування конфігураційного файлу – config.yaml.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку необхідне виконання наступних вимог:

– наявність GPU NVIDIA з підтримкою CUDA та мінімальним обсягом 4 ГБ VRAM. Це дозволить використовувати швидкість обчислень на GPU для виконання складних алгоритмів обробки зображень та аналізу контурів рельєфу. Мінімальний обсяг 4 ГБ VRAM дозволить ефективно опрацьовувати великі зображення високої роздільної здатності;

– достатній об’єм фізичного накопичувача. Зображення високої роздільної здатності можуть займати значний об’єм місця на диску, тому важливо мати достатньо вільного місця для їх зберігання та обробки;

– мінімум 8 ГБ RAM. Це дозволить запускати обчислення та обробку зображень у великій кількості, забезпечуючи при цьому високу продуктивність та швидкість роботи.

2.2 Завантаження застосунку

Для завантаження даного програмного забезпечення, користувачу необхідно перейти на сторінку репозиторію GitHub за наступним посиланням: https://github.com/nikita-filianin/Relief-Contours.

Перебуваючи на сторінці репозиторію, користувач має можливість завантажити архів з усіма необхідними програмними компонентами бібліотеки. Архіви для завантаження доступні у вкладці реліз-версій (“Releases”), що представляють собою останні стабільно-працюючі версії програмного забезпечення.

Після завантаження архіву, користувачу необхідно розпакувати його будь-яким зручним способом, після чого він матиме змогу відкрити проєкт у будь-якому зручному середовищі розробки (IDE).

2.3 Перевірка коректної роботи

Для локального розгортання та подальшого використання даної бібліотеки, користувачу необхідно мати встановлений інструмент для управління віртуальним середовищем Python у вигляді Conda. Для його встановлення необхідно зайти на офіційний сайт Conda.io та завантажити файл-інсталятор. Далі виконуються звичайні кроки встановлення, після чого все буде готовим до роботи.

Після встановлення Conda, користувач має відкрити проєкт бібліотеки у зручному для нього середовищі розробки. Далі необхідно відкрити термінал (командний рядок) та виконати наступні кроки:

– встановити PyTorch командою “conda install pytorch torchvision torchaudio pytorch-cuda=12.1 -c pytorch -c nvidia”;

– переконатись, що CUDA доступна (рисунок 2.1);

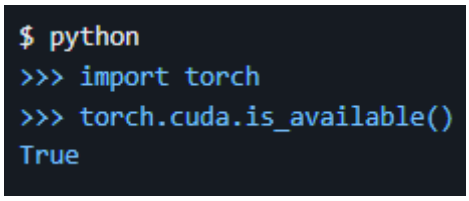


Рисунок 2.1 – Команди для перевірки доступності CUDA

– створити віртуальне середовище та встановити решту необхідних залежностей командою “conda env create -f contours.yml”;

– активувати створене віртуальне середовище командою “conda activate contours”.

Виконавши покроково вказаний алгоритм дій, користувач отримує налаштоване середовище для подальшого використання бібліотеки та її коректної роботи.

3 ВИКОНАННЯ ПРОГРАМИ

Перебуваючи в папці проєкту після активації середовища, термінал користувача має наступний вигляд (рисунок 3.1).

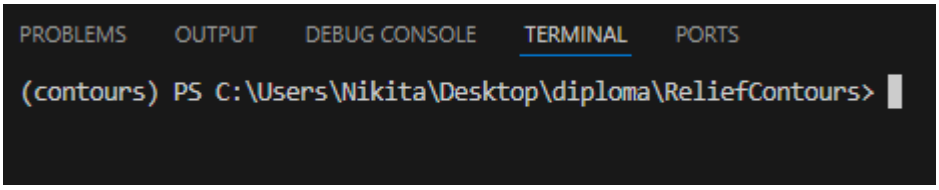


Рисунок 3.1 – Початковий вигляд терміналу

Від цього моменту, користувач має змогу використовувати CLI (Command Line Interface). Користувач має змогу переглянути список доступних для використання параметрів командою “python main.py --help” або “python main.py -h” (рисунок 3.2).

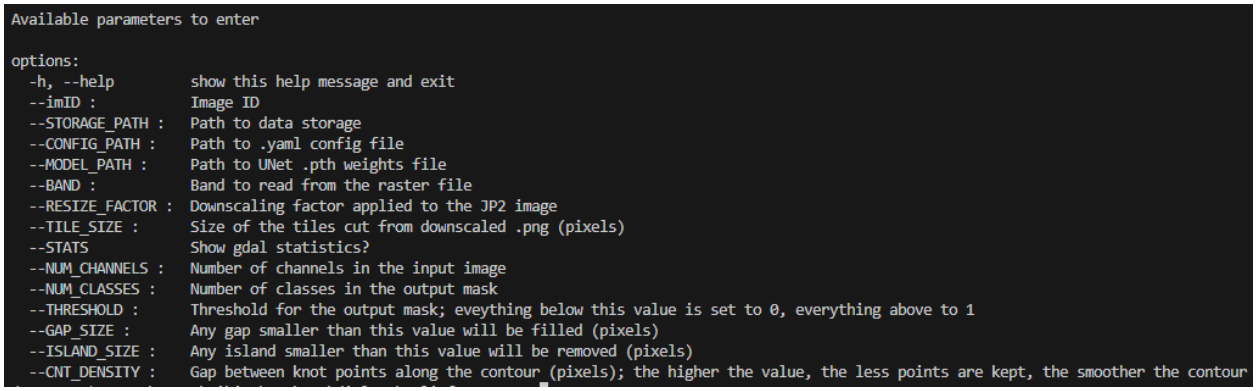


Рисунок 3.2 – Список доступних параметрів

Для вибору необхідного зображення на обробку, користувач має використати параметр “imID” для вказання ID зображення (рисунок 3.3).

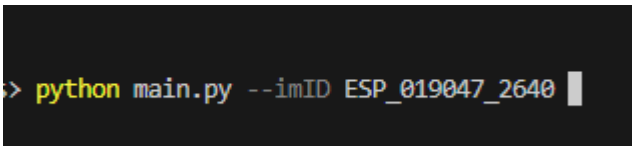


Рисунок 3.3 – Вказання ID зображення

Параметр “imID” обов'язковий для введення, оскільки без нього процес виконуватись не буде.

Користувач має змогу вказати шлях до необхідної директорії сховища даних, використовуючи параметр “STORAGE_PATH” (рисунок 3.4).

```
> python main.py --imID ESP_019047_2640 --STORAGE_PATH ../data2
```

Рисунок 3.4 – Вказання шляху до директорії сховища

Якщо користувач не задає цей параметр, то після першого запуску процесу обробки, буде автоматично створено директорію “data”, що буде представляти собою локальне сховище даних.

Користувач має змогу вказати шлях до конфігураційного файлу, використовуючи параметр “CONFIG_PATH” (рисунок 3.5).

```
> python main.py --imID ESP_019047_2640 --CONFIG_PATH Configs/config2.yaml
```

Рисунок 3.5 – Вказання шляху до конфігураційного файлу

Якщо користувач не задає цей параметр, тоді використовується конфігураційний файл, що вже наявний в бібліотеці.

Користувач має змогу вказати шлях до необхідної моделі нейромережі UNet, використовуючи параметр “MODEL_PATH” (рисунок 3.6).

```
> python main.py --imID ESP_019047_2640 --MODEL_PATH UNet/weights/UNet_model2.pth
```

Рисунок 3.6 – Вказання шляху до моделі

Якщо користувач не задає цей параметр, тоді використовується модель, що вже наявна в бібліотеці.

Користувач має змогу вказати номер каналу, який буде зчитано з растрового зображення, використовуючи параметр “BAND” (рисунок 3.7).

```
> python main.py --imID ESP_019047_2640 --BAND 1
```

Рисунок 3.7 – Вказання номеру каналу

Якщо користувач не задає цей параметр, тоді використовується значення, що прописане в стандартному конфігураційному файлі (config.yaml).

Користувач має змогу вказати коефіцієнт зменшення розміру (стиснення) зображення у форматі JP2, використовуючи параметр “RESIZE_FACTOR” (рисунок 3.8).

```
> python main.py --imID ESP_019047_2640 --RESIZE_FACTOR 0.2
```

Рисунок 3.8 – Вказання коефіцієнту стискання

Якщо користувач не задає цей параметр, тоді використовується значення, що прописане в стандартному конфігураційному файлі (config.yaml).

Користувач має змогу вказати розмір плиток, на які розподіляється стиснуте зображення, використовуючи параметр “TILE_SIZE” (рисунок 3.9).

```
> python main.py --imID ESP_019047_2640 --TILE_SIZE 512
```

Рисунок 3.9 – Вказання розмір плиток

Вказане користувачем значення має бути кратним числу 2, інакше буде виведене повідомлення про помилку. Також дуже рекомендується вказувати значення, яке використовувалось у тренувальному датасеті до моделі, інакше це може призвести до неочікуваних (некоректних) результатів обробки. Якщо користувач не задає цей параметр, тоді використовується значення, що прописане в стандартному конфігураційному файлі (config.yaml).

Користувач має змогу увімкнути виведення статистичних метаданих для обраного растрового зображення, використовуючи параметр “STATS” (рисунок 3.10).

```
> python main.py --imID ESP_019047_2640 --STATS
```

Рисунок 3.10 – Увімкнення відображення метаданих

Відображення метаданих відбувається під час процесу обробки зображення (рисунок 3.11).

```
Driver: JP2OpenJPEG/JPEG-2000 driver based on OpenJPEG library
Size is 104556 x 39522 x 1
Projection is PROJCS["Polar_Stereographic MARS",GEOGCS["GCS_MARS",DATUM["unnamed",SPHEROID["unnamed",
3376200,0]],PRIMEM["Reference meridian",0],UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","9122"]]
],PROJECTION["Polar_Stereographic"],PARAMETER["latitude_of_origin",90],PARAMETER["central_meridian",0
],PARAMETER["scale_factor",1],PARAMETER["false_easting",0],PARAMETER["false_northing",0],UNIT["metre"
,1],AXIS["Easting",SOUTH],AXIS["Northing",SOUTH]]
Origin = (-309011.125, 211215.375)
Pixel Size = (0.25, -0.25)
Band Type=UInt16
Min=0, Max=1023
```

Рисунок 3.11 – Відображення метаданих

Користувач має змогу вказати загальну кількість каналів у вхідному растровому зображенні, використовуючи параметр “NUM_CHANNELS” (рисунок 3.12).

```
> python main.py --imID ESP_019047_2640 --NUM_CHANNELS 1
```

Рисунок 3.12 – Вказання загальну кількість каналів у зображенні

Якщо користувач не задає цей параметр, тоді використовується значення, що прописане в стандартному конфігураційному файлі (config.yaml).

Користувач має змогу вказати кількість семантичних класів, які необхідно виявити на зображенні, використовуючи параметр “NUM_CLASSES” (рисунок 3.13).

```
> python main.py --imID ESP_019047_2640 --NUM_CLASSES 1
```

Рисунок 3.13 – Вказання кількості семантичних класів

Якщо користувач не задає цей параметр, тоді використовується значення, що прописане в стандартному конфігураційному файлі (config.yaml).

Користувач має змогу встановити поріг чутливості для конвертації матриці ймовірностей у бінарну маску, використовуючи параметр “THRESHOLD” (рисунок 3.14).

```
> python main.py --imID ESP_019047_2640 --THRESHOLD 0.5
```

Рисунок 3.14 – Встановлення порогу чутливості

Дане значення має бути вказане в діапазоні від 0 до 1, інакше буде виведено відповідне повідомлення про помилку. Якщо користувач не задає цей параметр, тоді використовується значення, що прописане в стандартному конфігураційному файлі (config.yaml).

Користувач має змогу встановити значення для заповнення прогалин, тобто менше якого усі прогалини у бінарній масці будуть заповнені пікселями. Для цього використовується параметр “GAP_SIZE” (рисунок 3.15).

```
> python main.py --imID ESP_019047_2640 --GAP_SIZE 50000
```

Рисунок 3.15 – Встановлення значення для заповнення прогалин

Якщо користувач не задає цей параметр, тоді використовується значення, що прописане в стандартному конфігураційному файлі (config.yaml).

Користувач має змогу встановити значення для видалення острівців, тобто менше якого усі об’єкти (острівці) у бінарній масці будуть видалені. Для цього використовується параметр “ISLAND_SIZE” (рисунок 3.16).

```
> python main.py --imID ESP_019047_2640 --ISLAND_SIZE 50000
```

Рисунок 3.16 – Встановлення значення для видалення острівців

Якщо користувач не задає цей параметр, тоді використовується значення, що прописане в стандартному конфігураційному файлі (config.yaml).

Користувач має змогу встановити значення відстані між точками, які формують контур, використовуючи параметр “CNT_DENSITY” (рисунок 3.17).

```
> python main.py --imID ESP_019047_2640 --CNT_DENSITY 200
```

Рисунок 3.17 – Встановлення значення відстані між точками контуру

Після вказання користувачем ID зображення та інших, за потреби, параметрів, він натискає кнопку “Enter” на клавіатурі для запуску процесу обробки. В терміналі буде виведено основну інформацію, щодо станів процесу обробки (рисунок 3.18).

```
(contours) PS C:\Users\Nikita\Desktop\diploma\ReliefContours> python main.py --imID ESP_019047_2640 --CNT_DENSITY 150
../data/images/JP2/ESP_019047_2640_RED.JP2 already exists
../data/images/png/ESP_019047_2640_RED.png already exists
Model loaded successfully
Processing Time: 19.0 seconds
(contours) PS C:\Users\Nikita\Desktop\diploma\ReliefContours>
```

Рисунок 3.18 – Приклад вигляду терміналу після обробки

Якщо користувач не бажає кожен раз вводити свої значення параметрів, використовуючи CLI, він також має змогу зберегти їх для подальшого свого використання шляхом редагування конфігураційного

файлу – config.yaml. Зберігши там зміни, задані в ньому параметри будуть використовуватись для подальших запусків процесу обробки (рисунок 3.19).

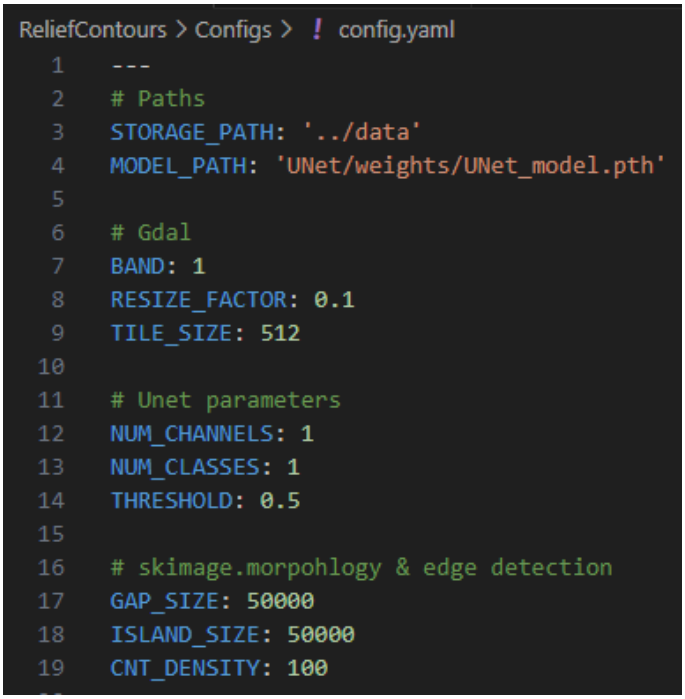


Рисунок 3.19 – Параметри, що задані у файлі config.yaml

Після завершення процесу обробки, про що свідчить виведення значення витраченого часу “Processing Time”, користувач має змогу переглянути отримані результати, що потрапляють в окрему папку всередині директорії сховища “data”, або того, що було вказане користувачем.

Результати представлені у вигляді NPY файлу, що зберігає список координат точок визначеного контуру та PNG файлу, що представляє собою зображення з графічно визначеним контуром відповідного рельєфу. Ці файли зберігаються у відповідних піддиректоріях всередині директорії “results” (рисунок 3.20).

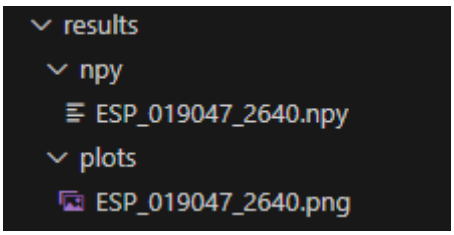


Рисунок 3.20 – Збережені файли результатів обробки

Користувач може переглянути результат у вигляді масиву координат точок контуру, що збережені в NPY файлі (рисунок 3.21).

```
[[ 1193  1159]
 [ 1210  1194]
 [ 1268  1224]
 [ 1196  1270]
 [ 1263  1336]
 [ 1305  1416]
 [ 1339  1479]
 [ 1385  1566]
 [ 1403  1660]
 [ 1470  1738]
 [ 1474  1803]
 [ 1428  1722]
 [ 1426  1779]
 [ 1486  1891]
 [ 1586  1986]
 [ 1654  1955]
 [ 1713  2034]
 [ 1760  2081]
 [ 1867  2153]
 [ 1958  2241]
 [ 2064  2331]
 [ 2126  2368]
 [ 2179  2370]
 [ 2219  2460]
 [ 2243  2566]
...
 [ 1537  1314]
 [ 1457  1254]
 [ 1350  1229]
 [ 1231  1173]]
```

Рисунок 3.21 – Приклад отриманого масиву координат точок контуру

Користувач також має змогу переглянути отриманий результат у вигляді PNG зображення з графічно визначеним контуром відповідного рельєфу (рисунок 3.22).

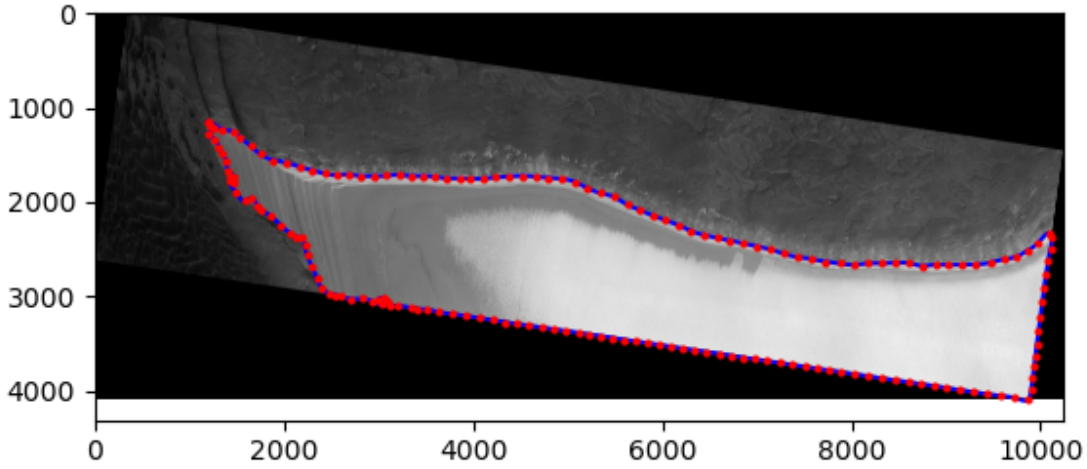


Рисунок 3.22 – Приклад отриманого зображення з визначеним контуром

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

БІБЛІОТЕКА ДЛЯ ВИЗНАЧЕННЯ КОНТУРІВ РЕЛЬЄФУ З СУПУТНИКОВИХ ФОТОГРАФІЙ

Графічний матеріал

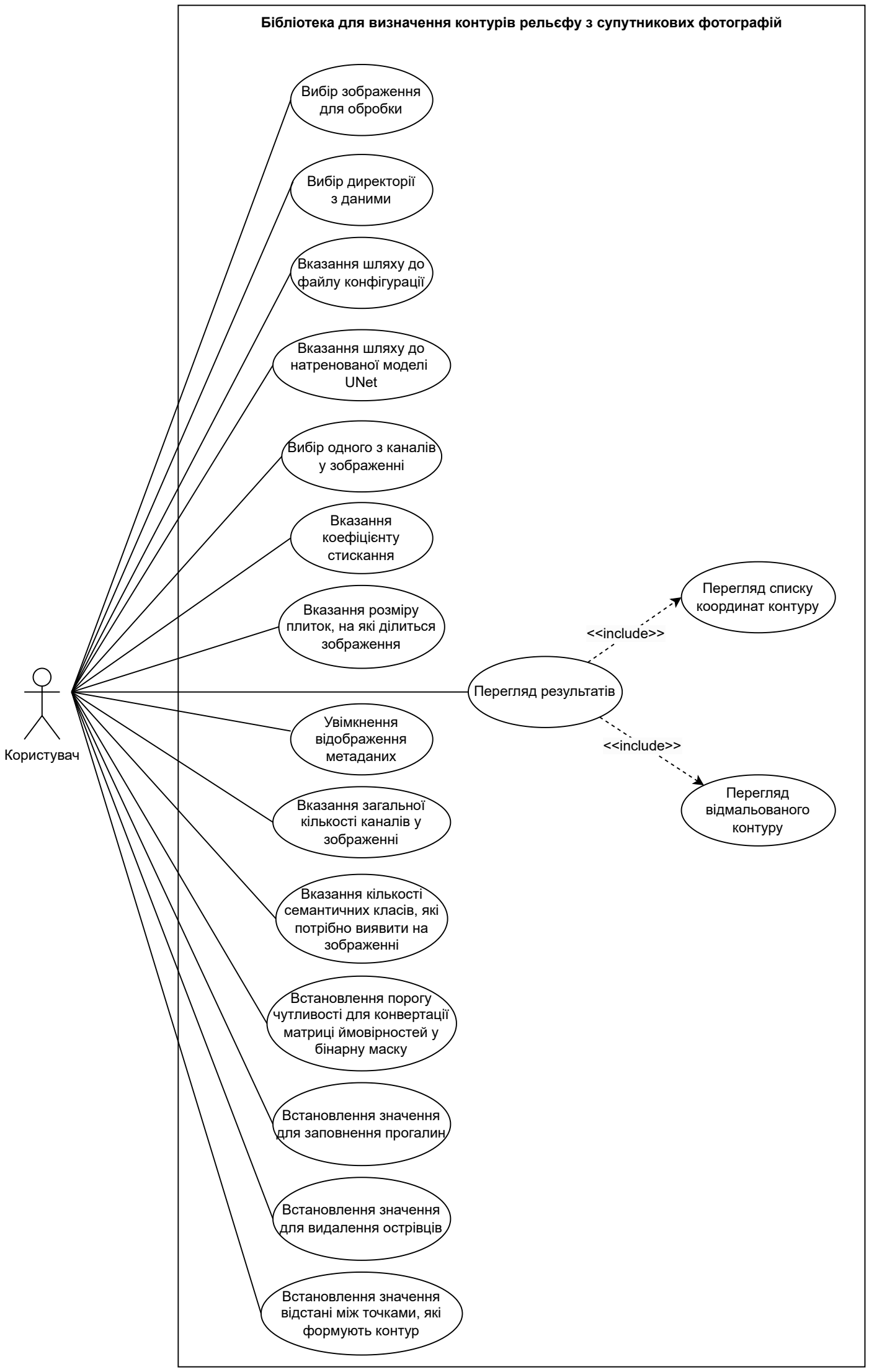
КПІ.ІТ-0321.045490.06.99

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олександр СТЕЛЬМАХ

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Нікіта ФІЛЯНІН

Київ – 2024

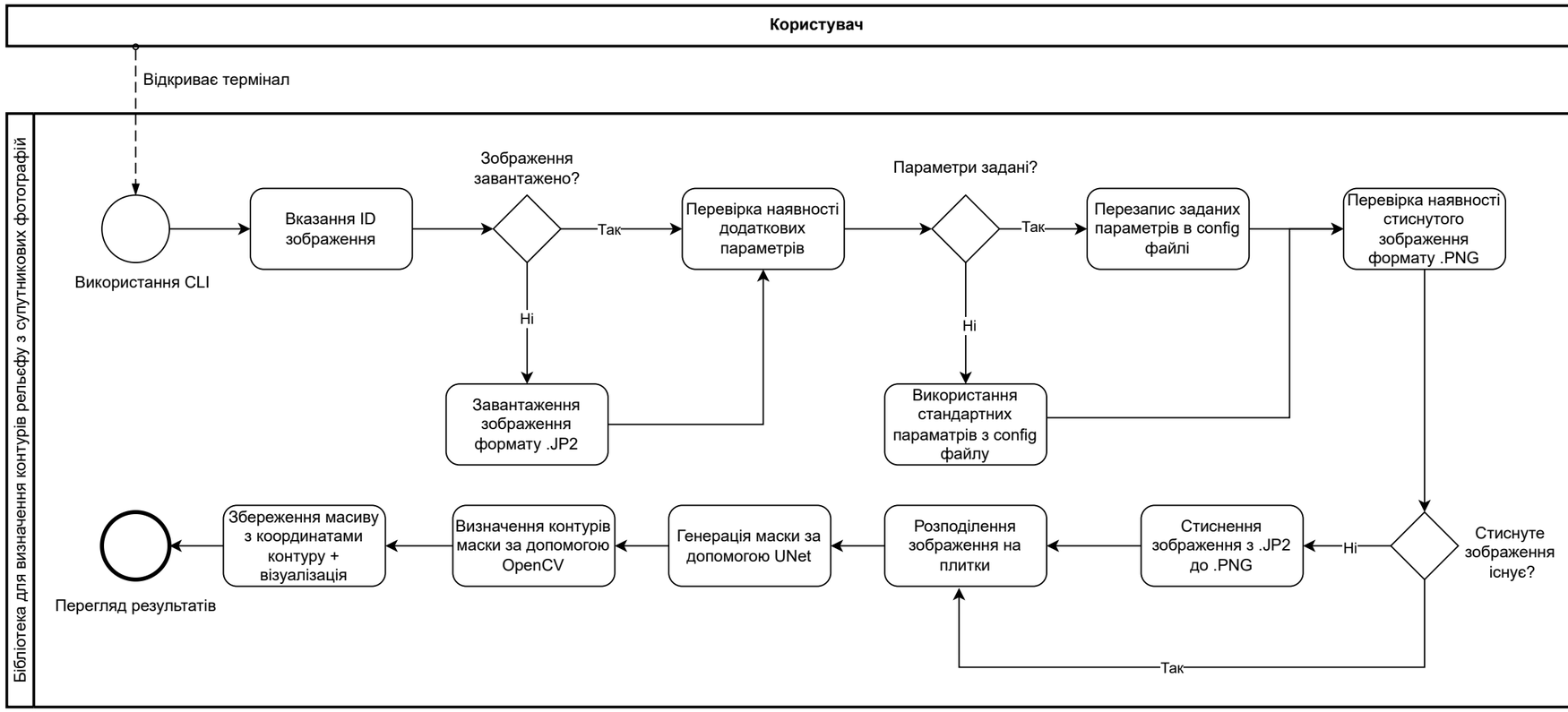


					КПІ.ІТ-0321.045490.06.99.ССВ
Зм.	Арк.	№ докум.	Підп.	Дата	
Розробив		Філянін Н.С.			Схема структурна варіантів використань
Перевірив		Стельмах О.П.			
Т. контр.					
Н. контр.		Вітковська І.І.			Бібліотека для визначення контурів рельєфу з супутникових фотографій
Затвердив		Жаріков Е.В.			

Лит.	Маса	Масштаб
Аркуш 1	Аркушів 4	

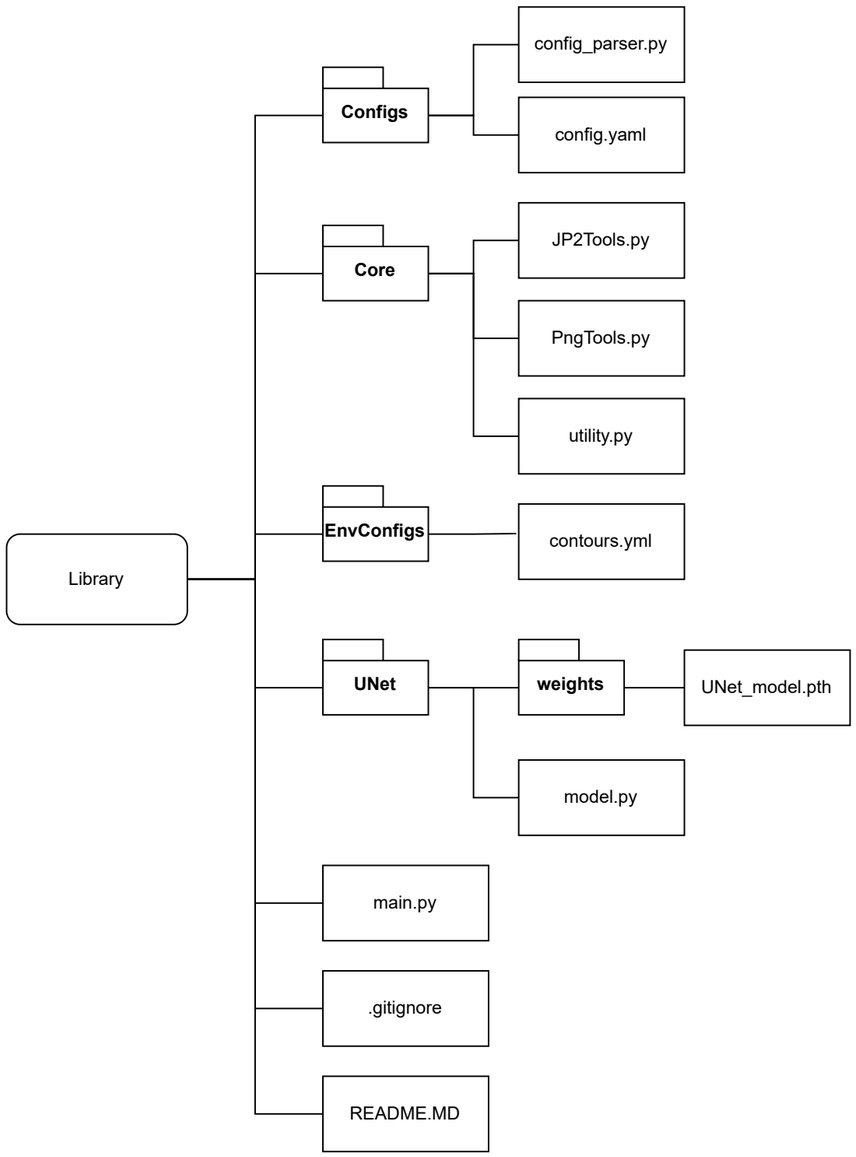
КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІТ-01

Обробка зображення



					КПІ.ІТ-0321.045490.06.99.ССД		
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна діяльності	Літера	Маса / Масштаб
Розробив		Філянін Н.С.					
Перевірив		Стельмах О.П.				Аркуш 2	Аркушів 4
Т. контр.							
Н. контр.		Вітковська І.І.			Бібліотека для визначення контурів рельєфу з супутникових фотографій	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-01	
Затвердив		Жаріков Е.В.					

Діаграма пакетів бібліотеки



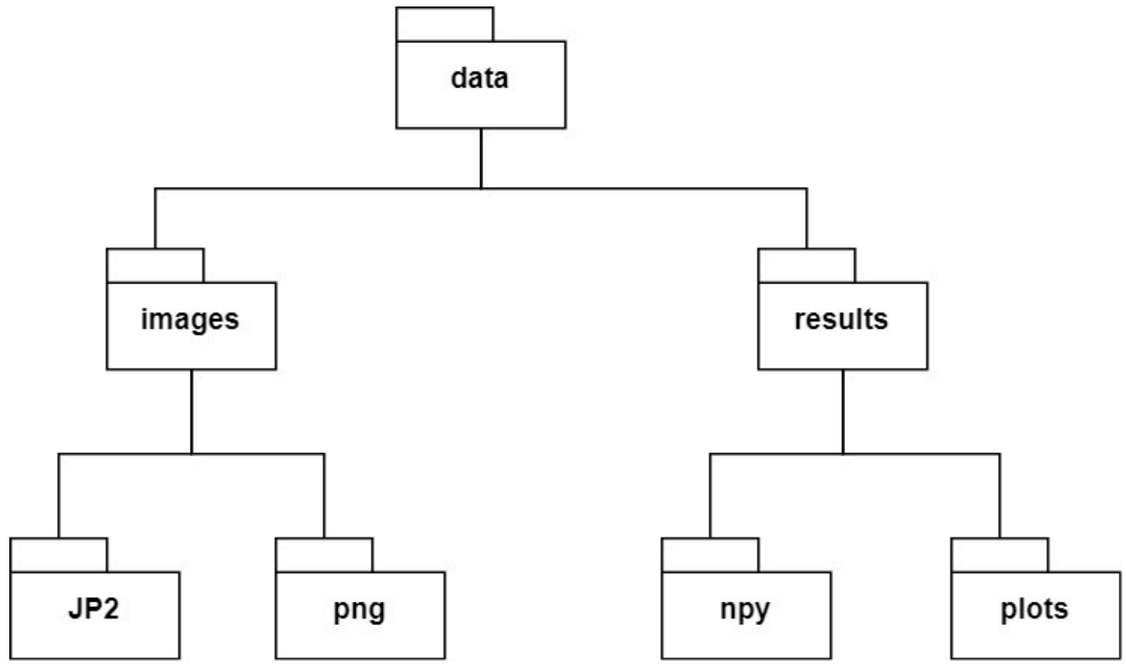
Демонстраційний плакат до дипломного проекту

Бібліотека для визначення контурів рельєфу з супутникових фотографій

Виконав студент гр. ІТ-01 Філянін Н.С.

Керівник Стельмах О.П.

Діаграма структури сховища



Демонстраційний плакат до дипломного проекту

Бібліотека для визначення контурів рельєфу з супутникових фотографій

Виконав студент гр. ІТ-01 Філянін Н.С.

Керівник Стельмах О.П.