ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Веб-ресурс для аналізу та класифікації земних матеріалів на основі гіперспектрального аналізу»

на здобуття освітнього ступеня бакалавра
зі спеціальності 122 Комп'ютерні науки
(код, найменування спеціальності)
освітньо-професійної програми Штучний інтелект
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Софія ХАЛАПОВА
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконала:
здобувачка вищої освіти Софія ХАЛАПОВА
група ШІД-41

Керівник: Андрій БОНДАРЧУК
науковий ступінь, вчене звання д.т.н., професор

Рецензент:
науковий ступінь, вчене звання _____
Ім'я, ПРІЗВИЩЕ

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Штучного інтелекту

Ступінь вищої освіти Бакалавр

Спеціальність Комп'ютерні науки

Освітньо-професійна програма Штучний інтелект

ЗАТВЕРДЖУЮ
Завідувач кафедрою Штучного інтелекту

_____ Ольга ЗІНЧЕНКО
«_____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Халаповій Софії Веніамінівні
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Веб-ресурс для аналізу та класифікації земних матеріалів на основі гіперспектрального аналізу

керівник кваліфікаційної роботи Андрій БОНДАРЧУК д.т.н., професор,
(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56

2. Строк подання кваліфікаційної роботи «23» травня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, відкриті набори даних HSI, технічні бібліотеки мови програмування Python.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження принципів класифікації RGB та HSI зображень
Аналіз технологій машинного навчання для задачі класифікації
Розробка системи класифікації та аналізу гіперспектральних зображень

5. Перелік графічного матеріалу: *презентація*
   1. Поняття Гіперспектральних зображень
   2. Архітектура існуючих методів класифікації машинного навчання
   3. Архітектура Згорткової Нейронної Мережі
   4. Аналіз та порівняння результатів класифікації

6. Дата видачі завдання«25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	25.02-04.03.25	Виконано
2	Вивчення поняття гіперспектральних зображень (HSI)	04.03-07.03.25	Виконано
3	Дослідження сфери застосування HSI	10.03-14.03.25	Виконано
4	Аналіз особливостей класифікації HSI	15.03-26.03.25	Виконано
5	Дослідження класифікаційних методів машинного навчання	27.03-04.04.25	Виконано
6	Застосування машинного навчання для класифікації об’єктів на HSI	05.04-05.05.25	Виконано
7	Оформлення роботи: вступ, висновки, реферат	06.05-13.05.25	Виконано
8	Розробка демонстраційних матеріалів	14.05-23.05.25	Виконано

Здобувачка вищої освіти _____ Софія ХАЛАПОВА
(підпис) *(Ім’я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Андрій БОНДАРЧУК
(підпис) *(Ім’я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 71 стор., 1 табл., 22 рис., 19 джерел.

Мета роботи – розробка інтерактивного веб-застосунку для аналізу гіперспектральних зображень.

Об'єкт дослідження – процес класифікації земних матеріалів на HSI.

Предмет дослідження – технології створення веб-ресурсу та класифікаційні методи машинного навчання.

Короткий зміст роботи: У роботі проведено огляд існуючих технологій розробки веб-ресурсу для класифікації предметів земної поверхні на основі гіперспектрального аналізу. Проведено порівняльний аналіз методів машинного навчання для задачі класифікації гіперспектральних даних.

Проведено аналіз особливостей будови, структури та спектрів HSI, а також їх впливу на якість класифікації. Проаналізовано вимоги до попередньої обробки даних, включаючи зменшення розмірності методом головних компонент (PCA), нормалізацію та аугментацію даних перед класифікацією. Проаналізовано підходи, які пропонує сучасне машинне навчання для роботи з HSI.

Використано мову програмування Python, фреймворк Flask для створення веб-застосунку, бібліотеки машинного навчання Scikit-learn, XGBoost та TensorFlow/Keras для реалізації моделей класифікації. Реалізовано інтерактивне завантаження наборів даних, вибір алгоритму класифікації, побудову графіків спектральних характеристик та класифікованих зображень.

КЛЮЧОВІ СЛОВА: HSI, КЛАСИФІКАЦІЯ, ЗОНДУВАННЯ ЗЕМЛІ, ЗГОРТКОВА НЕЙРОННА МЕРЕЖА, ВЕБ-ЗАСТОСУНОК.

ABSTRACT

Text part of the bachelor's qualification work: 71 pages, 22 pictures, 1 table, 19 sources.

Purpose of the research – the interactive web-application development got hyperspectral image analysis (HSI).

Object of the research – the process of classifying ground objects depicted on HSI.

Subject of the research – web-application development tools and machine learning classification methods.

Summary of the work: In presented work, existing web-development tools have been reviewed for the purpose of further Earth surface HSI object classification. A comparative analysis of machine learning methods for hyperspectral data classification was conducted.

The spectra and structural features of hyperspectral images were overseen. Also, during the work process, data preprocessing requirements were analyzed, including dimensionality reduction using Principal Component Analysis (PCA), normalization, and data augmentation before classification. HSI analysis approaches offered by modern machine learning were also examined in the contents of this work.

For the purpose of web-development, Python programming language along with the Flask framework have been used. Machine Learning libraries namely: Scikit-learn, XGBoost, TensorFlow/Keras were availed for implementing classification models. Interactive dataset uploading, classification algorithm selection, image spectral characteristics plotting and objects classification were implemented.

KEYWORDS: HSI, CLASSIFICATION, EARTH OBSERVATORY, CONVOLUTIONAL NEURAL NETWORK, WEB APPLICATION.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному світі швидкого розвитку та створення новітніх технологій аналіз зображень посідає важливе місце. У повсякденному житті людина стикається із зображеннями на постійній основі.

Обробка зображень застосовується не лише у побутових цілях, а також у галузях медичної діагностики, екологічного моніторингу земної поверхні, робототехніки, машинного зору тощо.

Для різних цілей і галузей використовуються різноманітні розширення, типи та види зображень. Сучасні технології дозволяють отримувати фото у різних спектральних діапазонах – від видимого світла, що сприймає людське око, до інфрачервоного, ультрафіолетового, радіохвиль тощо.

Гіперспектральна (HSI) зйомка знаходиться у діапазоні від ультрафіолету до ближнього інфрачервоного спектру (300–2500 нм). Гіперспектральні зображення застосовуються у дистанційному зондуванні Землі, виявленні мін та вибухонебезпечних предметів, екології та агрономії.

Галузі застосування гіперспектральної зйомки є актуальними, особливо враховуючи воєнні дії на території України для виявлення вибухонебезпечних предметів на поверхні Землі. Задля вдалого та плодотворного застосування HSI, інтеграція штучного інтелекту – це необхідність для вимог сучасного світу. Аналіз гіперспектральних зображень засобами ШІ дозволить точно визначати та аналізувати предмети на поверхні Землі та застосовувати отримані дані у розвитку вище зазначених галузей.

Мета дослідження – розробка інтерактивного веб-застосунку для аналізу гіперспектральних зображень, що дозволить будь-яким користувачам автоматично здійснювати спектральний аналіз та класифікувати об’єкти на зображенні за допомогою методів машинного навчання.

Об’єкт дослідження – процес класифікації земних матеріалів на основі гіперспектрального аналізу.

Предмет дослідження – сукупність технологій створення веб-ресурсу та методи машинного навчання для аналізу та класифікації об’єктів на гіперспектральних зображеннях.

Завдання дослідження:

- Проаналізувати предметну область: розглянути особливості гіперспектральної зйомки; методи класифікації машинного навчання; веб-застосунки із інтеграцією ШІ;
- Оглянути засоби реалізації веб-застосунку: розглянути особливості фреймворку, на якому буде написано застосунок; розглянути особливості створення бази даних на основі обраної мови програмування;
- Спроєктувати та реалізувати веб-застосунок з інтегруванням методів машинного навчання: побудувати діаграму класів; діаграму послідовностей; діаграму прецедентів; провести огляд основного механізму роботи;
- Розробити веб-застосунок з інтегрованою класифікацією об’єктів на гіперспектральних зображеннях.

Методи дослідження:

- Вивчення існуючих аналогів веб-ресурсу;
- Вивчення засобів машинного навчання для класифікації;
- Методи об’єктно-орієнтовного програмування;
- Тестування та відлагодження веб-застосунку;

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ІСНУЮЧИХ РІШЕНЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Опис предметної області проєкту

Гіперспектральна зйомка (HSI) дозволяє людям розширити свої можливості у військовій, аграрній та медичній сферах. Предметом гіперспектральної зйомки є зображення, що мають ширші спектральні характеристики ніж можуть передати звичайні RGB-зображення типів .jpg, .jpeg, .png тощо. На відміну від вище зазначених звичайних кольорових зображень, HSI містять сотні спектральних каналів, кожен з яких зберігає дані про довжину електромагнітних хвиль, що випромінюють предмети на зображенні.

Кожен піксель гіперспектральних зображень містить спектральні характеристики об’єктів, тобто як саме предмет на зображенні відбиває та поглинає світло на різних довжинах хвиль. Піксель HSI представляється як спектр - вектор значень відбиття або випромінювання на кожній з фіксованих довжин хвиль, а не як цифрове значення кольору. Тобто спектральна інформація у пікселях – це довжини електромагнітних хвиль (wavelength)

1.2 Огляд ресурсів для аналізу та класифікації зображень

Враховуючи стрімкий розвиток технологій штучного інтелекту у галузі аналізу, обробки, розпізнавання та класифікації зображення, не дивно, що було створено ресурси, які аналізують різноманітні типи зображень.

Одним з таких ресурсів є веб-застосунок Teachable Machine від корпорації Google, який спеціалізується на обробці різних типів медіа, включаючи зображення. Даний застосунок дозволяє користувачам без знань у програмуванні та штучному інтелекті створювати, тренувати та експортувати моделі машинного навчання для класифікації зображень, звуків чи відео.

Teachable Machine підтримує різноманітні формати завантаження моделей: TensorFlow.js (для вебзастосунків), TensorFlow Lite (для мобільних додатків), а також CoreML (для пристроїв Apple).

Даний ресурс має застосування у сфері освіти, досліджень, а також для демонстрації можливостей ШІ для звичайного користувача.

Інший схожий ресурс – IBM Watson Studio від відповідної компанії. Дана платформа – це більш комплексний і складний підхід до класифікації та аналізу зображень. Професійний веб-застосунок IBM Watson Studio призначений для data science та машинного навчання.

Перевага IBM Watson Studio – це гнучке налаштування моделей та вибір алгоритм навчання (логістична регресія, дерева рішень, нейронні мережі, тощо). Застосунок дозволяє також автоматизувати підготовку даних завдяки AutoAI або вручну нормалізувати дані у Jupyter Notebook на Python або R.

Оскільки ресурс інтегрований з IBM Cloud, готові моделі можна розгорнути як REST API у хмарі.

Також існує веб-застосунок від Microsoft Azure – Microsoft Custom Vision. Основний напрямок наведеного продукту від Microsoft – створення моделей комп’ютерного зору під особисті потреби та вимоги.

Платформа підтримує різноманітні формати для написання коду та має потужні інструменти для аналізу даних. Watson Studio дозволяє створювати власні моделі, завантажувати кастомні зображення, оцінювати точність моделей та експортувати їх для Azure, TensorFlow, ONNX, CoreML.

Із переваг даної програми: простий інтерфейс, інтеграція з Microsoft Azure та підтримка як класифікації так і розпізнавання об’єктів на зображеннях.

1.3 Актуальність розробки веб-програми

Вище зазначені ресурси для аналізу та обробки зображень широко застосовуються, але здебільшого орієнтовані на звичайні RGB-зображення або просту класифікацію об’єктів, не аналізуючи спектральні дані.

Через дану проблему на ринку залишається проблема створення спеціалізованих застосунків для роботи з гіперспектральними зображеннями. Такі веб-ресурси будуть корисні у аграрній промисловості, екологічному моніторингу, геології та медицині для людей, які не працюють на пряму зі штучним інтелектом, але потребують аналізу місцевості, рослинності, медичних знімків тощо.

Отже на ринку постає потреба впровадження застосунків, які дозволяють завантажувати власні HSI-датасети, обирати методи обробки й отримувати інтерпретовані результати у зручній формі. Такий підхід до створення застосунку для аналізу зображень забезпечить повноцінну підтримку багатовимірних спектральних даних, характерних для гіперспектральної зйомки.

Таким чином, розробка даного застосунку не лише дозволить розширити потенціал використання HSI-даних у зручному веб-інтерфейсі, але й зробить аналіз спектральності зображень доступним для ширшого кола дослідників, викладачів, студентів та інженерів. На відміну від вищезазначених платформ, веб-застосунок для класифікації HSI зображень направлений на вузькоспеціалізований, але критично важливий тип задач.

1.4 Застосування елементів ШІ для класифікації.

1.4.1 Поняття та роль ШІ в задачах класифікації.

Поняття Штучного Інтелекту існує ще з 50-х років 20 сторіччя, коли науковець Алан Тюрінг висунув ідею, що машину можна навчити думати. У 1950 році науковець створив спеціальний тест, що зараз має назву «Тест Тюрінга», який мав на меті визначити чи здатна машина проявляти інтелект,

думати. Перевірка полягала у тому що, якщо машина може переконати людину, що вона — також людина, то можна вважати, що машина проявляє інтелект.

Наразі, у 21 сторіччі, штучний інтелект є буденністю для більшості з нас. Люди звертаються до розумних чатів за порадами, проханнями проаналізувати папери, розповісти якусь цікаву людині інформацію тощо.

Але, звичайно, ШІ застосовують не лише звичайні люди. Для того, щоб користувачі отримували рекомендації згенеровані Штучним Інтелектом та використовували ШІ-асистентів, розробники та науковці створюють нові алгоритми та моделі штучного інтелекту, впроваджують засоби ШІ у застосунки, що ми використовуємо на щоденній основі.

Одним з напрямків впровадження ШІ є обробка зображень, а саме задачі класифікації. Штучний інтелект дозволяє автоматизувати процес розпізнавання об’єктів, ознак чи матеріалів на зображеннях, аналізуючи їх характеристики. Аналіз характеристик об’єктів на зображеннях здійснюється шляхом виявлення закономірностей у піксельних або спектральних даних, що дозволяє моделі з високою точністю визначати, до якого класу належить кожен елемент.

Для задач класифікації застосовується машинне навчання (ML) – підгалузь штучного інтелекту, яка направлена на імітування машинами того як навчається мозок людини. ML може виконувати завдання ШІ та покращувати продуктивність своїх моделей.

Класифікація це задача Машинного Навчання, що вимагає позначення мітками класів на основі їх характеристик. Характеристики – це числові або категорійні ознаки даних, які є вхідними даними для алгоритмів і використовуються для визначення закономірностей між даними та прогнозування класів. Метод класифікації застосовується у розпізнаванні зображень, фільтрації спаму, рекомендацій, медичній діагностиці тощо.

У процесі навчання модель "вчиться" на певних прикладах, тобто якщо в задачі класифікації на вхід подається зображення з міткою, до якого класу

воно належить, модель поступово знаходить такі риси зображень, які найкраще відрізняють один клас від іншого.

1.4.2 Попередня обробка та нормалізація RGB та HSI.

Для того, щоб модель якісно навчилась на вхідних даних та видала путні вихідні результати, потрібно попередньо обробити та почистити початкові зображення. Попередня обробка зображень критично впливає на точність та ефективність класифікації.

1.4.2.1 RGB-зображення

RGB – це найпоширеніший тип зображень, їх людина зустрічає щодня у повсякденному житті: роздруковані картинки, фото зроблені на телефон тощо. Вони складаються з трьох кольорових каналів: Red, Green, Blue. Для кожного каналу використовується своя колірна мапа (*colormap*). Піксель у зображеннях RGB містить у собі два послідовних значення індексу рядку та стовпцю з одновимірного векторного масиву, сукупність таких пікселів для red, green та blue утворюють одновимірний масив пікселів. Одновимірний масив пікселів перетворюється на двовимірну матрицю та візуалізується як RGB-зображення (рис 1.1).

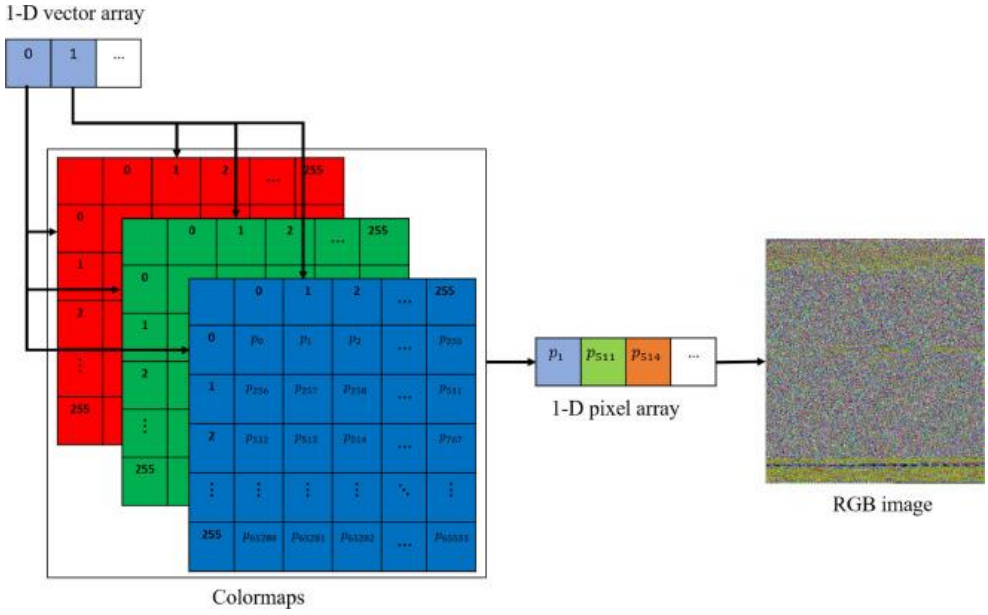


Рисунок 1.1 – Утворення RGB-зображень.

Попередня обробка RGB-зображень полягає у тому, щоб спочатку привести всі зображення до одного розміру. Значення пікселів у форматі RGB коливається на проміжку [0, 255], це пояснюється тим, що кожен кольоровий канал у цифрових зображеннях міститься у 8-бітових цілочисленних значеннях. 8-бітне число може бути представлено як 256 можливих значень у двійковій системі числення, тому значення пікселів коливаються у заданому діапазоні.

Наступним кроком є нормалізація піксельних значень, що передбачає перетворення пікселів з діапазону від 0 до 255 на [0, 1]. Таке зменшення розмірності пікселів допомагає знизити надмірність та підвищити загальну цілісність даних, що допомагає моделі «сконцентруватись» на оцінці головних характеристик.

Шуми на зображенні можуть перешкодити моделі якісно визначити класи та навчитись. Шуми – це випадкове коливання яскравості або кольору на зображеннях. Шум на зображенні може виникати через засвіти об’єктиву, зернистість плівки тощо. Шум може виглядати як зернисті або крапчасті плями, які зменшують якість фото. Задля зменшення природніх шумів, що виникають на будь-якому зображенні у великому або малому проявах, накладаються спеціальні фільтри або застосовуються алгоритми покращення контрастності.

Найпоширеніші фільтри:

- Фільтр середнього значення (заміняє кожне значення пікселя на середнє поміж сусідніми пікселями, тим самим вирівнюючи зображення та зменшуючи шум)
- Медіанний фільтр (обробляє кожен піксель, замінюючи його колір на медіанне значення серед усіх пікселів у квадратній області навколо обраного пікселя. Даний метод ефективно усуває дрібний шум на зображенні.)

- Фільтр Гауса (зменшує шум використовуючи розподіл Гауса для розмиття та згладжування шуму. Для кожного вхідного пікселя обчислюється нове значення, які обираються розподілом Гауса.)

Наступний крок попередньої обробки – аугментація даних.

Дана обробка полягає у збільшенні кількості даних для навчання, адже маючи більшу кількість вхідних параметрів, існує більша імовірність успішного навчання та класифікації. Аугментація або розширення даних – це технологія, яка використовується для штучного синтезу нових даних шляхом обертання, дзеркального відображення, зміни яскравості пікселів тощо.

Наступний пункт – це стандартизація. Стандартизація перетворює дані у послідовний формат, полегшуючи навчання для моделей ML. Стандартизація потрібна, щоб забезпечити рівномірність ознак, адже різні ознаки можуть мати різні діапазони значень, у випадку зображень – це інтенсивність кольору пікселя [0, 255]. Стандартизація робить усі ознаки рівними та безрозмірними, чим забезпечує більш рівне та точне розпізнавання та навчання.

1.4.2.2 Гіперспектральні зображення

Гіперспектральні зображення структурно відрізняються від звичайних. На відміну від трьох-канальних RGB-зображень HSI мають десятки або сотні каналів, кожен з яких відповідає за свій спектральний діапазон фото. Для обробки такого масштабного обсягу інформації потрібно багато місця та часу, тому зображення потрібно стиснути, щоб зберегти лише необхідні дані.

Попередня обробка гіперспектральних зображень також включає у себе видалення шумів, а саме шумових каналів. Шуми на гіперспектральних зображеннях виникають внаслідок атмосферних перешкод, змін температури, квантових перешкод, які виникають, коли аналоговий сигнал перетворюється на цифровий та фотонів, які погіршують якість гіперспектральних даних.

Зменшити шум можна завдяки використанню 3D моделей для декореляції шуму в просторовому та спектральному вимірах. Розрідження шуму здійснюється за рахунок того, що гіперспектральне зображення розглядається як тривимірний куб даних, де два виміри представляють просторову інформацію, а третій вимір спектральну інформацію.

Перевагами даного методу є ефективне використання просторових та спектральних даних, а також якісне збереження спектральної інформації, що є необхідною для аналізу та подальшої обробки зображень.

Інший спосіб – це зменшення шуму шляхом відкидання надлишкових спектральних смуг, тобто ми обираємо нижчу спектральну розмірність, щоб уникнути надлишковості.

Враховуючи, що HSI складаються із сотень, а іноді і з тисяч каналів, не всі вони є однаково корисними та інформативними для навчання. У спектрі можуть зустрічатись смуги з низьким рівнем сигналу, смуги з високим розшаруванням, що містять однакову інформацію тощо. Тобто метод полягає у зменшенні розмірності даних, усунення надлишкового спектрального шуму та підвищення точності класифікації за рахунок покращеного співвідношення сигналу до шуму.

Також на етапі видалення шуму варто видалити фон та виділити *ROI (Region Of Interest)* в HSI. *ROI* – це зразок набору даних, обраний для певного завдання. Область інтересу зазвичай використовується для виділення найважливіших ознак, на які має спиратись класифікація. Широко *ROI* використовується у медичному напрямку, для виділення зон пухлин або інших злоякісних утворень.

Наступний крок у обробці гіперспектральних зображень – це видалення «мертвих» пікселів. Даний тип пікселів виникає внаслідок можливих дефектів детекторів або датчиків, на які знято зображення. У ближньому інфрачервоному діапазоні, до якого належать гіперспектральні зображення, приблизно 1% пікселів можуть бути дефектними.

«Мертві» пікселі – це відсутні або нульові значення у зображенні, які спотворюють багатовимірні зображення, утворюючи хибні послідовності пікселів. Такі послідовності можуть заплутати модель машинного навчання, тому їх важливо вчасно визначити та замінити. Фільтрування «мертвих» пікселів також важливо і у звичайних зображеннях, але враховуючи масштабні розміри HSI, вони становлять більшу небезпеку точності моделі, оскільки суттєво спотворюють спектри.

«Мертві» пікселі визначають пороговою перевіркою по всіх каналах, тобто пікселі, які мають значення 0, позначаються як мертві. Також якщо спектр пікселів поруч мають приблизно однакове значення, тобто у них занадто низька дисперсія, такі пікселі – «мертві».

Після виявлення дефектних пікселів, вони замінюються на вирахувані значення відносно сусідніх пікселів, для цього використовують вище зазначені фільтри, такі як: медіанний, гаусівський тощо (рис. 1.2).

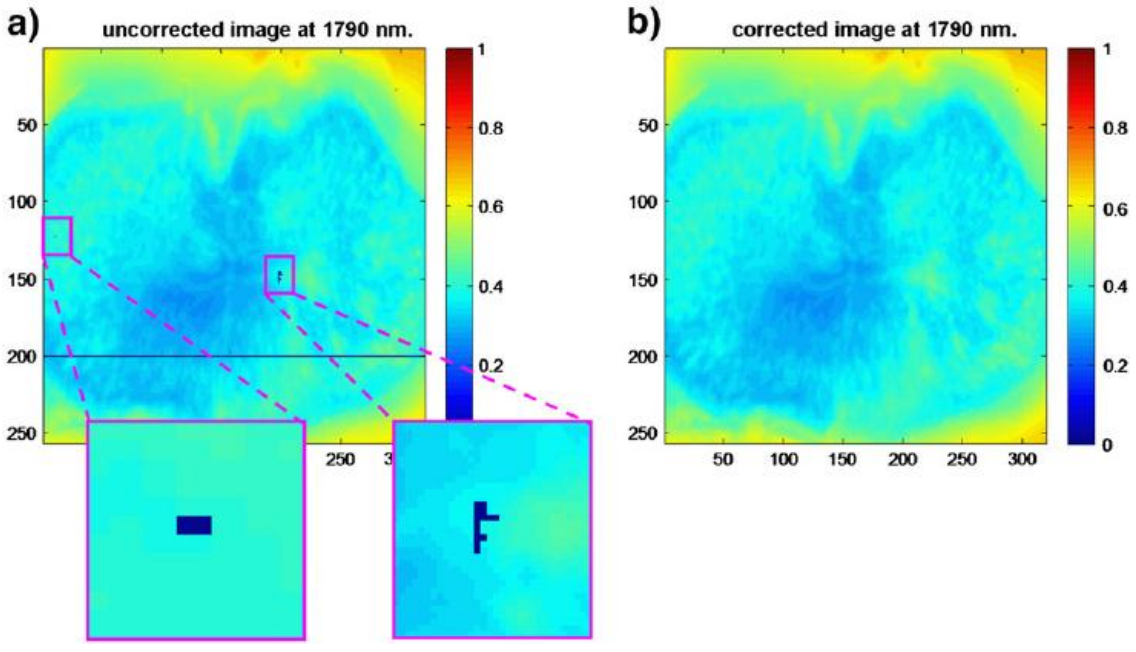


Рисунок 1.2 – Заміна «мертвих» пікселів.

Важливим етапом попередньої обробки HSI є нормалізація та стандартизація спектрів. Задля коректної класифікації дані мають містити

логічні закономірності, для цього всі спектри приводяться до одного масштабу.

Спочатку проводиться *Min-Max Scaling* – приведення значень у кожному каналі до діапазону [0, 1]. Так само як у RGB-зображеннях ми зменшуємо розмірність з [0, 255] на [0, 1] потрібно зробити і у гіперспектральних зображеннях, але у пікселях кожного спектру.

Далі проводиться *Z*-нормалізація, іншими словами стандартизація. Ця техніка перетворює дані на стандартний розподіл, який має однакові характеристики та масштаб, які модель легко сприймає та обробляє. Математично стандартизація полягає у перетворенні кожного значення ознаки, щоб у результаті було отримано середнє значення ознаки та стандартне відхилення.

У гіперспектральних зображеннях кожному пікселю належить певний спектр з N кількістю каналів (довжина хвилі – *wavelength*) і для кожного каналу обчислюються стандарте значення та відхилення по усьому зображенню або по раніше визначеному ROI. Після цього кожне значення спектра перетворюється за формулою:

$$z = \frac{x-\mu}{\sigma} \quad (1)$$

де:

$x -$ значення спектра;

μ – середнє значення;

σ – стандартне відхилення.

Як вже було згадано вище, важливим етапом попередньої обробки є зменшення розмірності даних. Для цієї задачі існують різноманітні алгоритми, такі як:

- *PCA* (метод головних компонентів, який передбачає виділення найголовніших компонентів спектра, зменшуючи кількість частотних каналів при мінімальній втраті інформації)

- *T-SNE* (T-розподілене вкладення стохастичної близькості призначене для зменшення розмірності шляхом відстеження подібності між точками даних у багатовимірному просторі. Задача алгоритму – перетворити дані з великою кількістю спектральних каналів у 2 або 3 виміри, щоб схожі об’єкти в початковому просторі залишались поруч у новому зменшеному просторі.)

Останній етап – виділення ознак. Після застосування всіх методів очищення даних та зменшення розмірності, спектри потрібно перетворити на набір ознак, по якому модель машинного навчання зможе визначати закономірності та класифікувати об’єкти. Для даної задачі будуються вектори ознак, що подаються на вхід у модель (рис. 1.3).

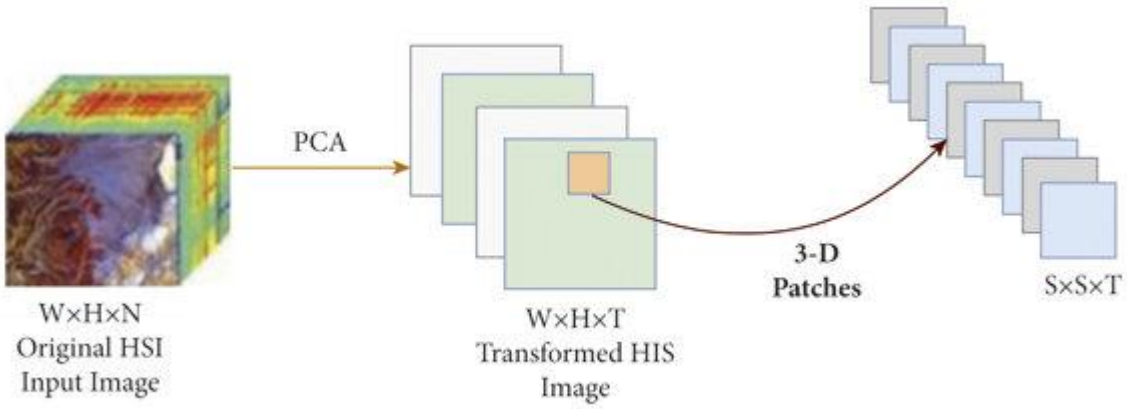


Рисунок 1.3 – Візуалізація зменшення даних методом PCA та перетворення спектрів на вектори ознак

1.4.2.3 Порівняння RGB та HSI

Отже, порівнюючи попередню обробку RGB-зображень та гіперспектральних зображень, можна зазначити, що попередня обробка останніх є досить комплексною і ускладненою.

Застосунки які спрямовані на обробку звичайних зображень ніколи не класифікують правильно об’єкти гіперспектральних зображень, що вкотре доводить важливість мати ресурс, спрямований конкретно на обробку та класифікацію гіперспектральних даних.

Таким чином, створення веб-програми, яка визнає специфіку попередньої обробки та застосування алгоритмів машинного навчання для обробки гіперспектральних зображень, є не лише актуальним, а й необхідним кроком у розвитку сучасних технологій роботи з зображеннями.

2 ЗАСОБИ ТА ТЕХНОЛОГІЇ СТВОРЕННЯ ВЕБ-РЕСУРСУ

2.1 Вибір технологій розробки веб-сайту

2.1.1 Flask

Для реалізації веб-застосунку було обрано фреймворк *Flask*. На відміну від більш багатофункціональних фреймворків, які надають широкий набір вбудованих інструментів (наприклад, *ORM* для роботи з базами даних, системи автентифікації, тощо). *Flask* надає лише базові, але необхідні компоненти для створення веб-додатків, залишаючи розробнику свободу вибору інструментів та бібліотек для реалізації решти функціоналу. Такий підхід є ідеальним для написання цього веб-застосунку, оскільки Flask дозволяє легко створити веб-застосунок, у який потім безпосередньо впроваджується логіка Машинного Навчання.

Flask включає у себе дві основні бібліотеки: *Werkzeug* та *Jinja2*. Це дві ключові бібліотеки фреймворку, які роблять використання Flask дуже гнучким.

Werkzeug – це багатофункціональна бібліотека, що використовує стандарт *Web Server Gateway Interface (WSGI). WSGI* контролює як веб-сервер взаємодіє з веб-застосунками, написаними на Python. У свій інтерфейс бібліотека включає засіб для відстеження та виправлення помилок застосунку. Також *WSGI* надає інтерфейс для взаємодії із заголовками HTTP, query parameters (дані, які передаються від клієнта, а саме веб-браузера, до сервера як частина URL-адреси), даними форм, завантаженими файлами та файлами cookie.

Бібліотека має набір інструментів для роботи безпосередньо з протоколом HTTP: кешування, інформація про клієнтів та користувачів, файли cookie, обробка завантажених файлів тощо.

Werkzeug також включає систему маршрутизації, призначену для зіставлення URL-адрес із кінцевими точками у застосунку та для генерації URL-адрес на основі кінцевих точок.

Багатопотоковий *WSGI*-сервер призначений для запуску та тестування застосунку без необхідності налаштовувати повноцінний веб-сервер. Даний функціонал являє собою простий веб-сервер, спеціально адаптований для зручного запуску та тестування *WSGI*-сумісних додатків (таких як додатки на Flask) безпосередньо на локальному комп'ютері.

Саме цей аспект багатопотоковості, у поєднанні з гнучкістю *Flask* у виборі додаткових бібліотек (для баз даних, автентифікації тощо), робить його чудовим вибором для невеликих веб-проєктів, як цей. Швидкість розробки, простота архітектури та відсутність потреби у складних вбудованих системах можливість швидко підняти робоче середовище та ефективно тестувати функціонал за допомогою *Werkzeug* є значною перевагою та вагомим аргументом на користь фреймворку *Flask*.

Друга не менш важлива бібліотека *Flask* – це Jinja2. Дана бібліотека спрямована на створення та візуалізацію веб-сторінок (HTML-документів), шляхом об'єднання структури шаблону на frontend з даними, що генеруються у вашому Python-додатку на backend. *Jinja2* є потужним та гнучким інструментом для обробки шаблонів, який *Flask* використовує за замовчуванням для формування відповідей користувачам.

Jinja2 виступає певним родом «перекладачем» між шаблонами сторінок та обробкою даних та логікою застосунку. Основне завдання цієї бібліотеки полягає у відділенні логіки представлення (як виглядає сторінка) від бізнес-логіки застосунку (як дані обробляються). Із функцій на backend, що регулює *Flask*, передаються дані або відповіді користувача у створені HTML-файли (шаблони), у яких використовуються спеціальні позначки Jinja2 для вставки динамічних даних ({{ змінна }}). Після даної передачі у веб-застосунку користувач буде бачити оброблені HTML-шаблони з даними, що були оброблені *Flask* та логікою на backend.

Перевагами jinja2 є гнучкість та зручність в управлінні даними у шаблонах (в файлах .html присутні такі структури даних як цикли, умови тощо). Також бібліотека підтримує ієрархію, тобто успадкування та перевикористання шаблонів, що дозволяє уникнути дублювання коду. Також jinja2 впливає на безпеку застосунку. Бібліотека забезпечує автоматичне екранування даних, що захищає від поширених небезпек як XSS-атаки.

2.1.2 SQLAlchemy

Зручним набором інструментів для створення та розгортання баз даних в Python є бібліотека SQLAlchemy.

SQLAlchemy яка слугує одночасно набором інструментів для роботи з SQL та Об'єктно-реляційним Мапером (ORM - Object Relational Mapper). Задача бібліотеки забезпечити гнучкість інтеграції та взаємодії впровадженої логіки в коді на Python з мовою програмування для керування реляційними базами даних SQL.

Бібліотека надає повний набір шаблонів для роботи зі стійкими даними (persistence patterns), які часто застосовуються в застосунках корпоративного рівня. Дані патерни лежать в основі SQLAlchemy та були розроблені з метою забезпечення максимально ефективного та високопродуктивного доступу до баз даних. При цьому вони адаптовані та представлені у вигляді простої та інтуїтивно зрозумілої мови предметної області в рамках Python, тобто для використання SQLAlchemy не потрібні інші середовища розробки. Ітрефейс взаємодії з даною бібліотекою полягає у створенні .py файлу та задання зв’зків, баз і таблиць на мові програмування Python після імпорту SQLAlchemy.

Бібліотека пропонує потужну мову виразів SQL (SQL Expression Language). Такий підхід дозволяє будувати SQL-запити програмно, використовуючи об'єкти та оператори Python, але зберігаючи синтаксис та семантику SQL як вже згадувалось раніше.

SQLAlchemy відома своєю різноманітністю, оскільки підтримує широкий спектр реляційних баз даних (PostgreSQL, MySQL, SQLite, Oracle, SQL Server тощо) та надає послідовний інтерфейс для роботи з ними.

2.1.3 SQLite

У конкретному випадку застосунку для даної кваліфікаційної роботи SQLAlchemy працює на основі реляційної бази даних SQLite.

SQLite – це компактна, вбудована система керування реляційними базами даних (СУБД). Вона відрізняється від інших СУБД, таких як MySQL або PostgreSQL, своєю легкістю впровадження, тобто тим, що вона вбудовується безпосередньо в застосунок, не вимагаючи окремого сервера баз даних.

Дана система так само легка та зручна як і бібліотека, що її використовує. Вона не вимагає великих ресурсів та характеристик «заліза» або окремих масштабних серверів. SQLite дозволяє продуктивно керувати даними, скорочуючи та забезпечуючи їхню цілісність.

Через свою простоту у використанні та легкість у вазі, SQLite не призначена для великих обсягів даних та інформації або застосунків, що потребують розподілених баз даних. Але для локального застосунку, який спрямовано на обгортання моделей машинного навчання більшого і не потрібно, тому SQLite є ідеальним рішенням для задач даної роботи.

Але не зважаючи на вище згаданий аспект, SQLite підтримує повний набір SQL-команд і не обмежує розробників ні у чому. Так само як у середовищі DBMS, наприклад, можна виконувати операції вставки, вибірки, оновлення та видалення даних з використанням SQL-запитів.

Також велика перевага даної системи – те, що SQLite вбудована в застосунок, тобто не потребує окремого сервера, як вже було зазначено, і тому адміністрування бази зводиться до мінімуму.

2.2 Вибір технологій розробки класифікаційної системи

Даний проєкт спрямований на розробку класифікаційної системи для розпізнавання об’єктів земної поверхні на гіперспектральних зображеннях.

Існує безліч алгоритмів машинного навчання для задачі класифікації, таких як: SVM, K-mean, наївний Байєс, метод найближчих сусідів (K-nearest neighbors - KNN), лінійна регресія, логістична регресія, дерево рішень тощо.

Усі ці підходи – це алгоритми машинного навчання з учителем направлені на класифікацію певної кількості визначених класів. Класифікація полягає у передбаченні категорії, до якої належить екземпляр на основі його характеристик.

Класифікація передбачає навчання на вже позначених наборах даних (labeled data), такі набори даних містять вже правильно класифіковані та позначені представники певних класів.

Після завершення етапів нормалізації даних та зменшення їх вимірності, які було описано в першому розділі, для задачі класифікації застосовується вище наведені алгоритми.

2.2.1 Метод класифікації XGBoost

Алгоритм градієнтного бустингу XGBoost – це бібіліотека розроблена для досягнення високої продуктивності, гнучкості та точності прогнозування класифікації. Модель XGBoost послідовно будує дерева рішень паралельно мінімізуючи помилки попередніх прогнозів (Додаток А, рис. А.1). На вхід передається набір ознак та відповідні їм цільові значення, які необхідно передбачити.

Початкове передбачення створюється для усіх зразків даних, після чого алгоритм аналізує помилки й використовує їх для покращення наступних дерев. На кожному кроці XGBoost оцінює якість окремих вузлів за допомогою формули (Similarity Score), що визначає, наскільки добре вузол корелюється з поточними помилками. Коли XGBoost робить прогноз для

нового зразка даних, він просто додає внески від усіх дерев рішень у моделі. Для розрахунку Similarity Score кожного вузла дерева рішень використовується формула (2) :

$$SS = \frac{(\sum residuals)^2}{\sum P_{i-1} \cdot (1 - P_{i-1}) + \lambda} \quad (2)$$

де:

SS – similarity score (оцінка подібності вузла);

$\sum$ *residuals* - сума похибок у вузлі;

P_{i-1} – ймовірність прогнозу помилок перед поділом;

$1 - P_{i-1}$ - дисперсія ймовірностей для бінарної класифікації;

λ – регуляризаційний параметр.

Після того, як алгоритм XGBoost визначив похибки передбачень, він шукає найкращий спосіб поділу даних для зменшення помилок. Для даної задачі алгоритм аналізує кожну вхідну ознаку, сортує всі її значення у зростаючому порядку та визначає можливі місця розбиття дерева. XGBoost аналізує дані, переглядаючи кожні дві сусідні точки у відсортованому списку. Він знаходить середнє значення між ними і використовує його як потенційний поріг для розгалуження дерева.

Далі алгоритм визначає, наскільки якісним буде це розбиття. Для кожного можливого розділення XGBoost обчислює, наскільки сильно воно покращить модель. Це відбувається за допомогою формули приросту (*Gain*), яка розраховує вигоду від створення нового відгалуження в дереві рішень. Алгоритм перевіряє всі можливі варіанти розділення даних і вибирає той, який дає найбільший приріст. Це допомагає максимально покращити точність прогнозів.

Щоб уникнути надмірного розгалуження дерева та втрати точності передбачення та "запам'ятовування" шуму у даних (що називається перенавчанням), XGBoost використовує усічення вузлів. Гіперпараметр γ задає мінімальний поріг для приросту. Якщо приріст батьківського вузла менший за γ, цей вузол видаляється. Такий підхід допомагає позбутися

вузлів, які не вносять значного внеску в навчання, і зменшує ризик перенавчання моделі.

Після того, як оптимальне розбиття знайдено, XGBoost обчислює вихідне значення для кожного вузла за формулою (3)

$$Output\ Value = \frac{\sum residuals}{\sum P_{i-1} \cdot (1 - P_{i-1}) + \lambda} \quad (3)$$

де:

$\sum residuals$ — сума залишкових похибок (різниця між передбаченими та фактичними значеннями);

P_{i-1} — ймовірність передбачення перед поточним поділом;

λ — регуляризаційний параметр, який допомагає уникнути перенавчання.

Для адаптації значень подальшої класифікації, XGBoost використовує перетворення початкової ймовірності (0.5) у логарифмічні шанси (log-odds) за формулою (4):

$$\log(odds) = log\left(\frac{P}{1-P}\right) \quad (4)$$

Отримане значення коригується, додаючи вихідне значення вузла, помножене на коефіцієнт навчання за формулою (5):

$$log(odds)new = log(odds)old + Output\ Value \times \eta \quad (5)$$

де:

η — коефіцієнт навчання (learning rate), який контролює швидкість оновлення прогнозу.

За кожну епоху розрахунку оновлене значення перетворюється на ймовірнісне за формулою (6):

$$P = \frac{1}{1 + e^{-\log(odds)}} \quad (6)$$

Завдяки цьому послідовному підходу XGBoost поступово покращує свої передбачення, будуючи послідовність дерев рішень, де кожне наступне дерево враховує і виправляє помилки попередніх моделей.

2.2.2 Метод класифікації Random Forest Classifier

Random Forest Classifier - це ансамблевий алгоритм машинного навчання. Даний метод будує та навчається із застосуванням балансування класів, що забезпечує рівномірний розподіл даних поміж класами. Алгоритм Random Forest використовує множину дерев рішень для створення кінцевого вирішального прогнозу (рис. 2.2).

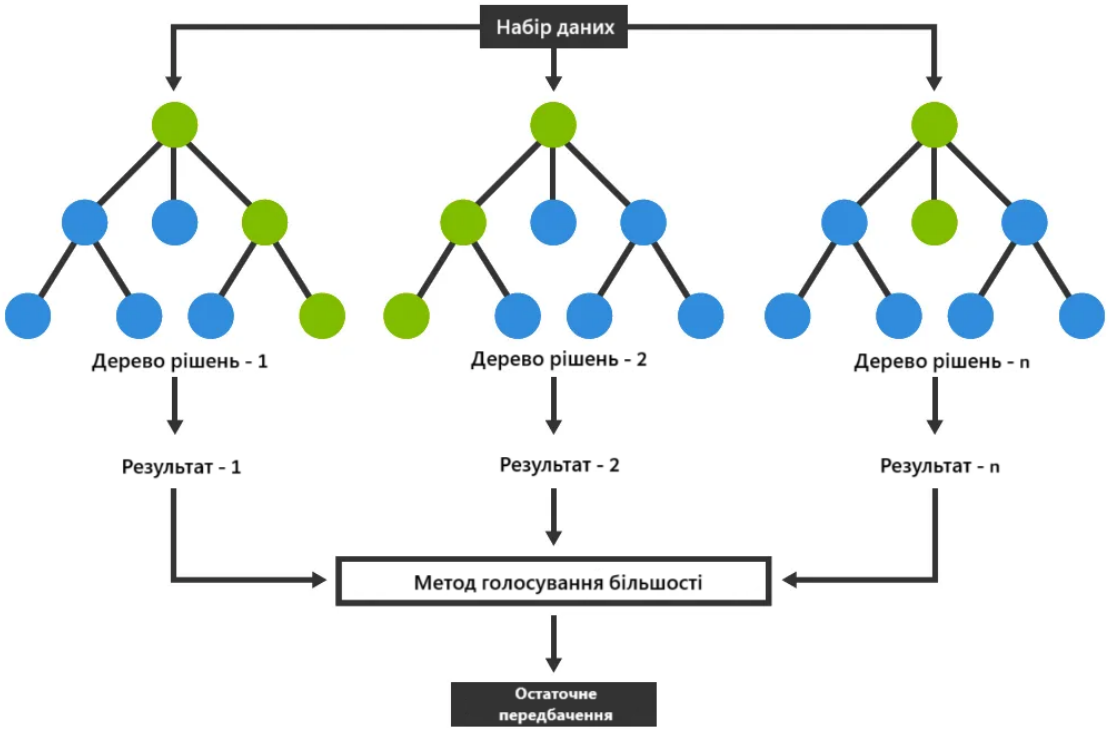


Рисунок 2.2 – Схематичне зображення роботи алгоритму Random Forest

Random Forest будує "ліс" незалежних дерев рішень. Кожне окреме дерево в лісі навчається на випадковій підмножині даних.

Побудова лісу починається зі створення унікальних навчальних наборів для кожного дерева. Це відбувається за допомогою методу Bootstrap Sampling. Тобто для кожного дерева випадково відбирається частина даних і ознак, створюючи таким чином індивідуальну вибірку для навчання.

Кожне дерево навчається на своїй унікальній підвибірці, що не тільки робить його особливим, а й підвищує загальну стійкість моделі. Під час навчання дерева використовують критерії розбиття, щоб визначити,

наскільки якісно можна розділити дані в кожному вузлі. Популярними критеріями є індекс Джині або ентропія. Вони допомагають обрати найкращі вузли для подальшого розгалуження дерева за формулою (7):

$$G = 1 - \sum_{i=1}^{n} p_i^2 \quad (7)$$

де:

p_i^2 – ймовірність належності випадкового елемента до і-го класу;

n – загальна кількість класів.

Чим менше значення індексу Джині, тим "чистішим" вважається вузол, що означає ефективніше розділення даних.

Важливо, що дерева рішень у Random Forest зазвичай не піддаються значному "обрізанню" (pruning) після їх побудови, на відміну від окремих дерев рішень. Їхня глибина може бути досить великою, оскільки остаточний прогноз базується на об'єднанні результатів багатьох дерев.

Після того, як усі дерева побудовані та навчені, відбувається об'єднання їхніх передбачень. Кожне дерево робить свій власний прогноз щодо класу певного зразка. Остаточне передбачення визначається методом голосування більшості (*Majority Voting*). Клас, який отримав найбільшу кількість "голосів" від усіх дерев, стає фінальним прогнозом. Це можна виразити формулою (8):

$$Y_{final} = \arg\max_{k} \sum_{i=1}^{N} I(y_i = k) \quad (8)$$

де:

Y_{final} – остаточний прогноз,

N – загальна кількість дерев лісу

$I(y_i = k)$ – індикаторна функція, яка дорівнює 1, якщо дерево і класифікує об'єкт як клас k, 0 в іншому випадку.

Використання безлічі випадкових дерев допомагає усунути шум у даних і значно підвищити точність моделі. Такий підхід ефективно запобігає перенавчанню, роблячи модель більш стійкою до неточностей у даних і

ефективною навіть при роботі зі складними, високорозмірними та нерозподіленими даними.

2.2.3 Метод класифікації K-Nearest

Метод K-найближчих сусідів це ще один алгоритм машинного навчання з учителем для задач класифікації, а також регресії. Суть даного алгоритму полягає у тому, розпізнати об’єкти, які знаходяться поруч одне з одним, тобто знайти «сусідів». Метод передбачає, що предмети, які лежать поруч одне з одним ймовірно мають схожі значення, ознаки, характеристики, параметри, тобто відносяться до одного класу.

У назві алгоритму стоїть параметр K, що визначає кількість «сусідів» з навчального набору даних, які будуть враховані при визначенні до якого класу відноситься новий, невідомий об'єкт. У задачі класифікації після знаходження K-найближчих сусідів довкола невідомого предмету, він отримує найпоширеніший клас серед сусідів шляхом Majority Voting (голосування більшості).

Задавати дуже низьке значення параметру K не рекомендується, адже такий підхід може призвести до впливу на модель шумів та локальних особливостей даних. Якщо єдиний знайдений сусід біля невідомого об’єкту – це викид, шум або належить до інакшого класу, то передбачення моделі виявиться хибним.

Але занадто високе значення K може нашкодити моделі також. Через низьку кількість сусідів модель потерпає від шумів, чого не стається при високому значенню K. При більшій вибірці сусідів вплив викидів на зображенні згладжується та шуми не так затьмарюють класифікацію. Однак завелике значення K призводить до недонавчання, особливо якщо класи перетинаються і щільно перемішані між собою. Модель може почати ігнорувати локальні закономірності, оскільки вибірка занадто велика. Оскільки велика вибірка включає і сусідів досить далеко від невідомого об’єкта, то він помилково може бути класифікований як вони.

Для визначення «сусідів» алгоритм розраховує відстань між невідомим об’єктом і заданою кількістю сусідів. Для цієї задачі найчастіше використовуються такі математичні формули як: Евклідова Відстань (9) або Мангеттенська метрика (10).

$$Euclidian\ Distance = \sqrt{\sum_{j=1}^{n} (X_1^j - X_2^j)^2} \tag{9}$$

Евклідова дистанція вимірює найкоротшу пряму відстань між двома точками у Евклідовому просторі.

Мангеттенська відстань вимірює відстань між вдома точками уздовж осей під прямим кутом.

$$Manhattan\ Distance = \left| \sum_{j=1}^{n} X_1^j - X_2^j \right| \tag{10}$$

Якщо дивитись графічно на відстань, виміряну цими формулами, то Евклідів шлях буде прямою, а Мангеттенський шлях буде нагадувати міські квартали або сітку.

Найближчими сусідами визначаються K точок з найменшою обчисленою відстанню до невідомого об’єкта (рис. 2.3).

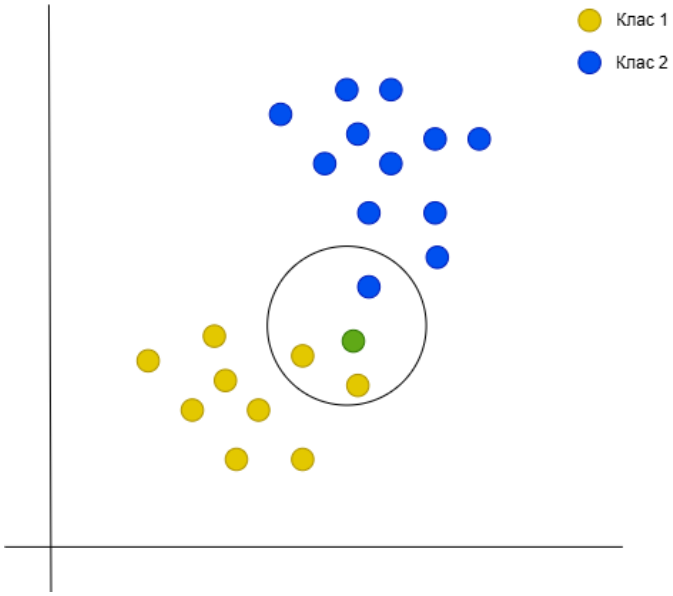


Рисунок 2.3 – Графічна візуалізація найближчих сусідів навколо невідомої точки зі значенням K=3

Після знаходження найближчих сусідів потрібно віднести невідому точку до певного класу. Вибір класу, якому варто присвоїти невідомий об’єкт вибирається вище згаданим методом голосування - *Majority Voting*.

Даний метод присвоює невідомій точці клас, який зустрічається найчастіше серед найближчих сусідів.

2.2.4 Метод класифікації Logistic Regression

Задача логістичної регресії передбачити вірогідність, що екземпляр належить до певного класу.

Класична логістична регресія, біноміальна, спрямована на бінарну класифікацію, тобто на визначення класів між двома об’єктами. Спрямованість на бінарну класифікацію полягає у використанні сигмоїдної функції логістичною регресією. Дана функція визначає імовірності, які за правилом лежать у діапазоні від 0 до 1, тобто чи належить цей об’єкт до певного класу – 1, чи не належить – 0, для чого зазвичай застосовується порогове значення (наприклад, 0.5).

Сигмоїдна функція формує S-подібну криву (рис. 2.4), яка може приймати будь-яке дійсне число та відображати його в вище зазначеному діапазоні. Якщо крива прямує до + нескінченності, прогнозований показник прийме значення 1, а якщо крива прямує до - нескінченності, прогнозований показник прийме значення 0.

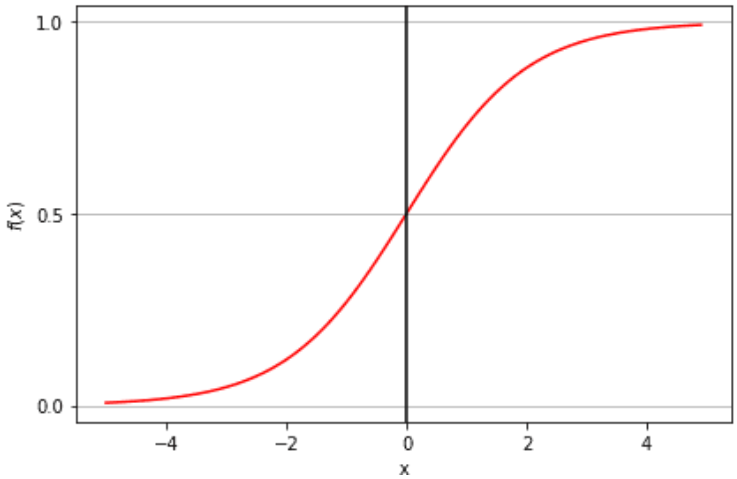


Рисунок 2.4 – Графік сигмоїдної функції

Логістична регресія спочатку обчислює лінійну комбінацію вхідних ознак точки (11):

$$z = w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n \quad (11)$$

де:

w – коєфіцієнти моделі, що навчаються на основі тренувальних даних.

Навчання логістичної регресії полягає у знаходженні оптимального набору коефіцієнтів, вищезазначених ваг, які найкраще описують зв'язок між вхідними ознаками та цільовою змінною. Визначення ваг потрібно для оптимізації, оскільки вони мають сприяти мінімізації помилок моделі. Початкові коєфіцієнти моделі – випадкові, а з часом модель їх оновлює.

Після логістична регресія застосовує до результату лінійної комбінації вхідних ознак сигмоїдну функцію (12).

$$f(x) = \frac{1}{1+e^{-x}} \quad (12)$$

де:

x – це лінійна комбінація вхідних ознак об'єкта з відповідними вагами:

Дана функція «стискає» будь-яке вхідне число (від мінус нескінченності до плюс нескінченності) до значення, яке завжди знаходиться в діапазоні від 0 до 1, тобто імовірність (13). Саме аспект оперування імовірністю лежить у причині бінарної класифікації, бо імовірність має два значення (p та q).

$$0 < p(X) < 1 \quad (13)$$

Для подальшої оптимізації, на основі прогнозованої імовірності потрібно визначити наскільки прогнози моделі відрізняються від справжніх значень у навчальному наборі, тобто алгоритм намагатиметься зробити усі значення більше порогової змінної максимально наближеними до 1, а нижче порогу – до 0, такі перетворення забезпечує функція втрат (*Log-Loss*) (14):

$$LL = \sum_{i=0}^{n} -(y_i * \log(p_i) + (1 - y_i) * \log(1 - p_i)) \quad (14)$$

де:

n – кількість зразків даних,

y_i – істинний клас i-го об’єкта в навчальному наборі,

p_i – прогнозована моделлю імовірність, що об’єкт належить до класу.

Після обчислень втрат застосовується оптимізаційний алгоритм, який покроково коригує значення коефіцієнтів моделі, щоб функція втрат досягла свого мінімуму. Найширше застосовують для оптимізації коефіцієнтів логістичної регресії «Градієнтний Спуск» (*Gradient Descent*) або його варіанти.

Градієнт – це вектор, який спрямовано у бік максимальної зміни функції (у нашому випадку, функції втрат). Градієнтний Спуск використовує цей вектор для визначення напрямку, у якому втрати зменшуються якнайшвидше, і покроково рухається в цьому напрямку, щоб наблизитись до мінімуму функції втрат (*Log-Loss*).

Також існує мультиноміальна логістична регресія. Даний підхід спрямований на класифікацію багатьох класів, оскільки замість сигмоїдної функції використовується функція *Softmax* (15), що приймає вектор значень, по одному від кожного класу, та перетворює його на вектор імовірностей, сума яких дорівнює 1. Кожен елемент цього вектору являє собою ймовірність належності об'єкта до відповідного класу.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{15}$$

де:

$\vec{z}$ – вхідний вектор значень,

e^{z_i} – стандартна експоненціальна функція для вхідного вектору,

K – кількість класів,

e^{z_j} – стандартна експоненціальна функція для вихідного вектору.

2.2.5 Метод класифікації Support Vector Machine (SVM)

SVM або метод опорних векторів класифікує дані, знаходячи оптимальну пряму або гіперплощину, яка забезпечує найбільшу відстань між кожним класом у N-вимірному просторі (рис. 2.5).

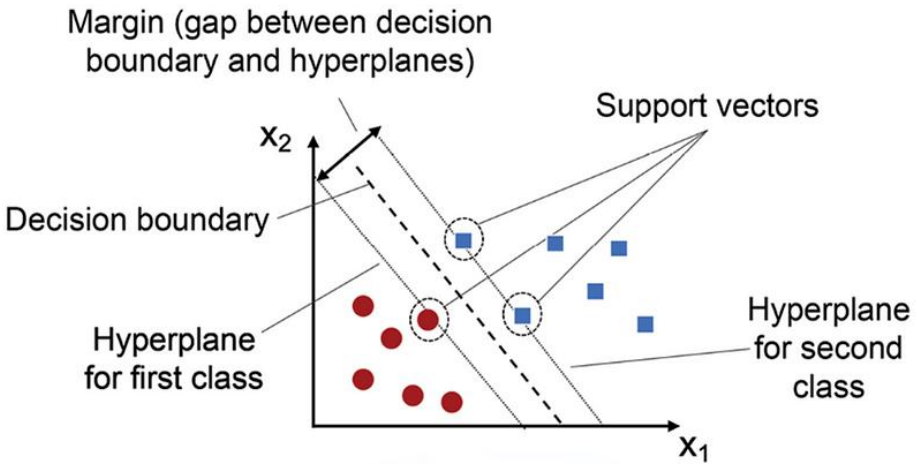


Рисунок 2.5 – Гіперплощина, яка найкраще розподіляє два класи, максимізуючи запас між ними

Відстань між гіперплощиною і найближчими точками даних з кожного класу називають запасом в SVM. Метою алгоритму є максимізація запасу одночасно мінімізуючи помилки класифікації. Більший запас вказує на вищий рівень точності класифікації, оскільки він означає наявність ширшого простору між гіперплощиною та найближчими точками даних кожного класу. Запас SVM - це показник того, наскільки добре розділені класи в просторі ознак.

SVM визначає найбільший запас для побудови більш надійної, стійкої та ефективної моделі, яка добре працюватиме на «брудних» багатовимірних даних. Пошуки запасу між прямою (гіперплощиною) та точками потрібні для узагальнення даних, щоб подивитись на повноцінний обсяг даних та наявних класів.

Даний алгоритм визначає, що найкраща для класифікації гіперплощина, яка знаходиться найбільш далеко від найближчих точок класів, оскільки така площина є менш чутливою до шумів, викидів або перенавчання і краще охоплює повноцінну структуру даних.

У реальних даних завжди є певний шум або природні варіації. Якщо гіперплощина проходить дуже близько до точок даних, які класифікують, невелика зміна в ознаках нової точки (через шум) може легко призвести до

неправильної класифікації. Тому через таке узагальнення і підхід SVM гарно працює на багатовимірних даних, таких як гіперспектральні зображення.

Після визначення гіперплощини та повноцінного аналізу даних, визначаються опорні вектори. Опорні вектори - це точки навчального набору, що є найближчими до визначеної прямої (гіперплощини) і лежать за межами запасу (рис. 2.6).

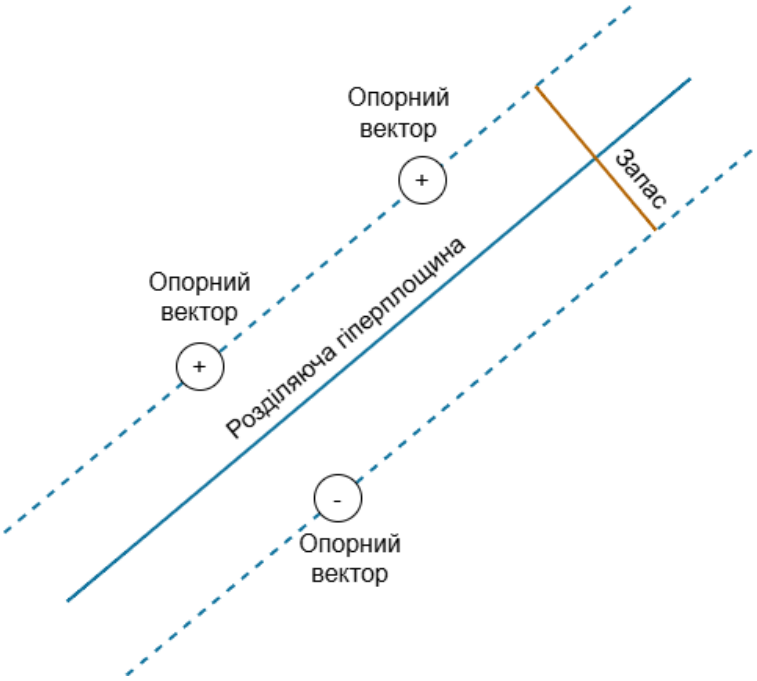


Рисунок 2.6 – Опорні вектори

Точки даних опорні вектори визначають положення та орієнтацію гіперплощини і тому мають значний вплив на точність класифікації SVM. Опорні вектори використовуються для обчислення запасу, який є відстанню між гіперплощиною та найближчими точками даних з кожного класу.

2.2.6 Метод класифікації Naive Bayes

В основу алгоритму Naïve Bayes входить однойменна теорема, яка полягає у припущенні, що усі змінні набору даних є незалежними одна від одної. Теорема базується на понятті умовної імовірності, тобто на імовірності появи першої події, якщо друга подія вже відбулась. Класифікація нового об’єкта здійснюється шляхом вибору того класу, який має вищу умовну імовірність.

Теорема Байєса пояснює зв’язок між умовною та оберненою умовною ймовірностями (16). У контексті класифікації вона виглядає так

$$\mathrm{P}(A|B) = \frac{P(B|A)*P(A)}{P(B)} \quad (16)$$

де:

A і B – це деякі події,

$P(A)$ – вірогідність правдивості гіпотези А (незважаючи на дані),

$P(B)$ – імовірність ознак (незважаючи на гіпотезу),

$P(A|B)$ – вірогідність правдивості гіпотези А, враховуючи дані В,

$P(B|A)$ – імовірність ознак В, якщо гіпотеза А правдива.

Класифікація наївним Байєсом лежить у виборі класу, який максимізує $P(B|A) * P(A)$ При обчисленні $P(B|A)$ – імовірності наявності всіх ознак одночасно за правдивості гіпотези, робиться вище згадане припущення, що кожна ознака незалежна від інших, якщо вже відомо клас об’єкта. Також кожній ознаці задається певна вага задля більшого або меншого впливу на навчання.

Такий підхід перетворює складну задачу оцінки багатовимірної умовної імовірності на задачу оцінки одновимірних умовних імовірностей для кожної ознаки окремо.

Таким чином перший крок навчання – це обчислення необхідних ймовірностей на основі тренувального набору даних. Для кожного класу обчислюється апріорна імовірність класу

Наступним кроком розраховуються умовні імовірності ознак за класом. Спосіб обчислення залежить від типу ознак та обраного типу Наївного Байєса (Мультиноміальний, Гаусівський, Бернуллівський). Наприклад, для дискретних ознак розраховується, скільки разів певне значення зустрічалося в об'єктах даного класу, і ділиться на загальну кількість об'єктів цього класу. Для неперервних ознак оцінюються параметри розподілу значень ознаки в межах наданого класу.

Далі модель починає прогнозувати значення, для чого використовується рівняння Наївного Байєса (17). Дана формула розраховує апостеріорні ймовірності для кожного можливого класу, використовуючи вивчені на етапі навчання імовірності.

$$\mathrm{P}(A|B) = \frac{1}{P(B)} * (P(A) * \prod_{i=1}^{n} P(B_i|A) \quad (17)$$

Значення за наведеною формулою обчислюються для всіх заданих K-класів. Тобто рівняння визначає значення, яке є пропорційним до апостеріорної ймовірності належності об'єкта до певного класу за наявності його ознак. Надалі об'єкту присвоюється той клас, для якого обчислене значення є найбільшим.

3 РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ ІЗ ЗАСТОСУВАННЯМ ШІ

3.1 Вибір мови програмування для проєктування застосунку

Для розробки проєкту було обрано мову програмування Python. Python – це високорівнева мова, що використовується у галузях веб-розробки, аналітики даних, машинного навчання, системного адміністрування, автоматизації тестування тощо.

Python – це об’єктно-орієнтована мова, що використовує об’єкти з чітко визначеними зв’язками.

Дана мова була обрана на основі її численних переваг, таких як:

Великою перевагою Python на фоні інших мов як Python, C++, C# є динамічна типізація. Дана характеристика робить Python, зручним і легким у користуванні. Динамічна типізація – це особливість мов програмування, у яких тип змінної визначається під час виконання програми, а не під час компіляції. У таким мова значення мають типи, а змінні – ні, тому кожна змінна може містити значення будь-якого можливого типу.

Python знають за його швидкість і зручність розробки. Зрозумілий і лаконічний синтаксис Python спрощує як вивчення, так і процес розробки: існує кілька стандартних бібліотек та розширень у PyPI (бібліотека пакетів Python). Крім того, Python має фреймворки, що надають попередньо написані компоненти (наприклад, *Flask* для веб-розробки загаданий вище).

Універсальність Python полягає також у його платформо-незалежності. Тобто Python дозволяє запускати один і той же код на різних операційних системах, таких як Windows, Linux або macOS. Портативність досягається завдяки байткоду та PVM (віртуальній машині Python), які слугують посередником між кодом, що пише програміст і процесором, що виконує програму. Python також може працювати в парі з іншими мовами, використовуючи такі розширення, як Cython для C, Gython для Go, Jython для Java та IronPython для .Net. Це дозволяє розробникам поєднувати мови та розширювати можливості своїх програм, беручи з усіх мов найкраще.

І остання, але не менш важлива перевага, чому було обрано Python – це безкоштовний та відкритий код. Python зроблений із ліцензією з відкритим кодом, що робить його безкоштовним для використання та розповсюдження.

3.2 Аналіз бібліотек обраної мови програмування

3.2.1 OS

Модуль OS дозволяє виконувати операції взаємодії з операційною системою, а саме виконання операцій з файлами (створення, видалення, перейменування), директоріями, поточним файловим середовищем. Даний модуль надає інформацію про середовище виконання файлів, тобто доступ до змінних оточення, інформацію про користувача та операційну систему.

OS забезпечує вище згадану платформо-незалежність Python, оскільки надає єдиний інтерфейс для виконання операцій незалежно від типу операційної системи. Конкретний модуль абстрагує відмінності між операційними системами, що дозволяє писати ко не беручи до уваги особливості окремих ОС.

3.2.2 Keras

Модуль Keras - це високорівневий API для задач глибокого навчання, який працює на базі бібліотеки низькорівневого машинного навчання TensorFlow. API (Application Programming Interface) - це набір інструментів та правил, які дозволяють різним програмам взаємодіяти між собою. API виступає посередником, який визначає, як програми можуть та обмінюватись даними, функціональністю та виконувати різноманітні задачі, використовуючи функції іншої програми.

Keras побудований для зручного використання розробниками, він спрощує процес побудови нейронних мереж, розподіляючи побудову нейронної мережі на абстракті процеси – модулі. Модулі складаються із

заданих розробником шарів нейронної мережі, оптимізаційних шарів та функцій втрат. Як вже було зазначено раніше, Keras працює на основі низькорівневих бібліотек як TensorFlow, Theano, CNTK.

Основна ідея створення Keras поверх інших бібліотек – зручний користувацький інтерфейс та полегшений алгоритм створення та запуску нейронних мереж. Дана бібліотека підтримує декілька способів побудувати модель для навчання нейронних мереж: послідовний (*sequential API*), функціональний (*functional API*) та модель підкласів (*model subclassing*) (рис. 3.1).

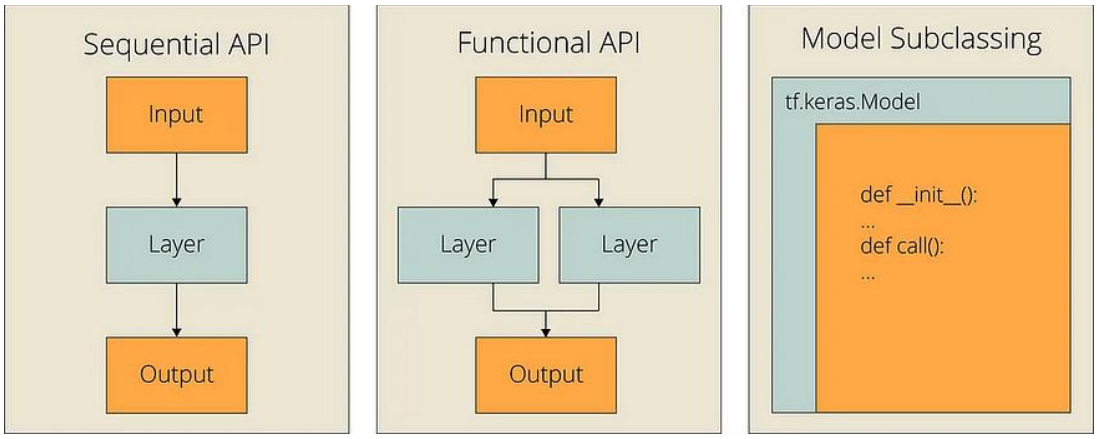


Рисунок 3.1 Keras API’s

Послідовне API найпоширеніше для використання, оскільки воно має один вхід і один вихід та використовується для побудови простих моделей, де шари розташовуються послідовно, один за одним. Враховуючи прямолінійну архітектуру Sequential API, багаторівневі та складні моделі проблематично реалізувати через даний підхід.

Для більш комплексних моделей та нейронних мереж використовується Functional API. Даний спосіб дозволяє декілька входів та виходів, а також покращеними та більш складними зв’язками між шарами моделі. Functional API є одночасно простим у використанні та достатньо гнучким для побудови різноманітних архітектур глибокого навчання.

Останній підхід Model Subclassing вимагає впровадження моделі з нуля, наслідуючи клас tf.keras.Model. На відміну від попередньо написаних API як Sequential та Functional модель підкласів потрібно імплементувати

вручну. Такий підхід надає максимально гнучкий інтерфейс розробки моделей нейронних мереж з динамічною особливою поведінкою.

3.2.3 NumPy

Наступна бібліотека застосована для написання даного проєкту – NumPy. Даний модуль для наукових обчислень та перетворень. Бібліотека базується на об’єкті ndarray – об’єкт, який може містити великі багатовимірні масиви та матриці. Такі масиви зберігають дані одного типу, що надає можливість ефективно зберігати та оперувати великими наборами однорідних даних. Ndarray може представляти будь-який тип даних від одновимірних, двовимірних масивів до складних матриць, та n-вимірних масивів, також відомих як тензори.

Бібліотека пропонує широку вибірку високорівневих математичних операцій та функцій для роботи з масивами тензорами, такі як: базові математичні операції (складання, віднімання, множення та ділення даних у масивах), задачі лінійної алгебри (множення, віднімання та інвертація матриць), трансформації Фур’є, тригонометричні та статистичні операції.

NumPy є основою для багатьох інших популярних бібліотек на Python для аналізу даних, таких як Pandas, SciPy, Matplotlib та scikit-learn. Саме цей аспект робить NumPy ключовим елементом для розробки високоефективних та масштабованих рішень машинного навчання, які завжди включають у себе роботу з великими об’ємами багатовимірних даних, як от гіперспектральні зображення. Швидкість виконання операцій у NumPy досягається завдяки реалізації значної частини коду на мовах C та Fortran, що забезпечує значно кращу продуктивність порівняно зі стандартними списками Python. Це дозволяє ефективно працювати з великими обсягами даних, що є критично важливим для даного проєкту.

3.2.4 SciPy

SciPy – це бібліотека з відкритим кодом на Python, яка використовується для наукових і технічних обчислень, особливо для задач машинного навчання. Вона побудована на основі NumPy і надає додаткові можливості для виконання таких задач, як: оптимізація даних, обробка зображень на основі тензорів оброблених NumPy, статистичні розрахунки тощо.

Перевага використання SciPy у її багатофункціональності, бібліотека є комплексною та покриває алгоритми, створення моделей та інші маніпуляції з обчислювальними даними, які не надають Matplotlib, Pandas та NumPy, наприклад. SciPy складається з певних модулів, кожен з яких спеціалізується на конкретних функціях для різних наукових та інженерних задач. Основні модулі SciPy включають: scipy.optimize, scipy.integrate, scipy.linalg, scipy.stats, scipy.fft, scipy.spatial, scipy.signal, scipy.ndimage.

Optimize – для задач оптимізації (включає обчислювальні методи для мінімізації та максимізації функцій, пошуку коренів рівнянь, задач лінійного програмування);

Integrate – чисельне інтегрування та розв’язання диференціальних рівнянь;

Linalg – більш розширені операції лінійної алгебри, ніж є у NumPy. Модуль включає в себе розв'язання лінійних систем, обчисленням визначників, матричні перетворення (LU-розклад, QR-розклад, розклад за сингулярними значеннями (SVD));

Stats – великий набір інструментів для статистичних обчислень, побудов гіпотез та розподілів імовірностей (містить реалізації численних одновимірних та багатовимірних розподілів з процедурами для оцінювання параметрів, симуляції даних та обчислення статистичних показників: середнє значення, дисперсія тощо);

Fft – застосовується для швидкого перетворення Фур’є для обробки сигналів. Підтримує обчислення як прямих, так і обернених перетворень для дійсних та комплексних наборів даних.;

Spatial – містить інструменти для роботи з просторовими даними та виконання геометричних операцій, обчислення відстаней (функції для розрахунку відстаней між точками та кластерами, включаючи Евклідову відстань та інші метрики.);

Signal – аналіз та обробка сигналів (фільтрація, згортка, спектральний аналіз). Очищення сигналів від шуму та виділення певних частотних компонентів;

Ndimage – обробка та аналіз багатовимірних зображень, що базуються переважно на n-вимірних масивах NumPy (фільтрація зображень від шумів, функції для згортки та кореляції).

3.2.5 Matplotlib

Дана бібліотека призначена для побудови графіків, діаграм та візуалізації метрик та параметрів моделей. Дана бібліотека тісно інтегрована з NumPy, що дозволяє з обчислених даних, що зберігаються у вигляді тензорів, легко зобразити результати на графіках та діаграмах. Бібліотека підтримує велику кількість типів візуалізації, таких як: лінійні графіки, гістограми, стовпчасті та кругові діаграми, 3D-графіки, теплові мапи.

Бібліотека дозволяє детально налаштовувати кожен елемент графіку: кольори ліній, маркери, шрифти, заголовки, підписи осей, розміри графіків та багато іншого.

3.2.6 Sklearn

Sklearn або scikit-learn – широко застосована бібліотека з відкритим доступом, що надає широкий вибір алгоритмів та інструментів для завдань машинного навчання та data science. Scikit-learn побудований на основі вищезгаданої бібліотеки NumPy для ефективної роботи з багатовимірними

числовими масивами даних. Також ця бібліотека є надбудовою над SciPy та Matplotlib.

Scikit-learn включає в себе широкий спектр алгоритмів для різних завдань машинного навчання, таких як класифікація, регресія, кластеризація та зниження розмірності. Доступні алгоритми включають лінійні моделі, дерева рішень, методи опорних векторів, ансамблеві методи тощо.

Бібліотека надає різноманітні інструменти для попередньої обробки даних, такі як StandardScaler для нормалізації даних, OneHotEncoder для кодування категоріальних змінних, та Imputer для заповнення відсутніх значень.

Бібліотека також включає функції для розбиття даних на навчальні та тестові набори за допомогою train_test_split, що дозволяє оцінювати ефективність моделей на небачених даних.

Бібліотека також надає інструменти для оцінки ефективності моделей, такі як кросс-валідація, метрики якості (accuracy_score, precision_score, recall_score тощо).

3.2.7 Seaborn

Наступна бібліотека для розгляду – Seaborn. Даний модуль призначений для візуалізації статистичних графіків та діаграм. Seaborn надає безліч функцій для створення візуально красивих, лаконічних та приємних графів. Графіки у Seaborn переважно використовують для візуалізації залежностей між змінними або класами у наборах даних.

Бібліотека надає інструменти для побудови багатьох адаптивних графіків, основні з яких наступні:

- Графіки залежностей (зображення взаємозв’язків між змінними);
- Категоріальні графіки (візуалізація та робота з категоріальними змінними, тобто дискретні, ті, які приймають значення з обмеженого набору категорій, груп або міток, не числові значення);

- Графіки розподілу (такі графи візуалізують форму, центр, розмах і наявність викидів у одновимірних (розподіл однієї змінної) та двовимірних (розподіл двох змінних одночасно) наборах даних);
- Мульти-графічні сітки (використовують для відображення декількох екземплярів одного і того самого графіка на різних підмножинах датасету).

Seaborn побудований на основі Matplotlib. Бібліотека дозволяє використовувати всі функції Matplotlib для налаштування графіків Seaborn (додавання назв осей, регулювання розмірів шрифтів, збереження графіків у різних форматах). Seaborn має легший і більш інтуїтивний підхід до створення складних графіків, які в Matplotlib вимагали б значно більше розробки, автоматично застосовуючи привабливі стилі та кольорові палітри

Також Seaborn тісно інтегрований з бібліотекою Pandas, що робить його дуже зручним для роботи з даними у форматі DataFrame. Модуль дозволяє передавати DataFrame у функції, вказувати назви стовпців для осей (колір, розмір тощо). Це значно спрощує синтаксис коду порівняно з Matplotlib.

Ще одна перевага Seaborn перед Matplotlib – це матричні графіки, які візуалізують взаємозв’язки між змінними у великих наборах даних. Наприклад *heatmaps* (теплові мапи) дозволяють показувати кореляційні матриці, що виділяють сильні та слабкі зв’язки між числовими змінними.

3.3 Опис коду та архітектури веб-застосунку

Даний застосунок розроблено на фреймворку *Flask* на мові програмування Python, що згадані та описані раніше. Мета застосунку – надати користувачу інтерфейс для взаємодії із гіперспектральними зображеннями, а саме для аналізу спектрів та класифікації об’єктів на земній поверхні. Для класифікації було застосовано декілька алгоритмів машинного

навчання: XGBoost, Random Forest, Logistic Regression, K-nearest neighbors та розроблену власноруч згорткову нейронну мережу.

Застосунок містить базу даних, інтерфейс особистих кабінетів користувачів та безпосередню роботу з гіперспектральними проєктами

Конфігурація застосунку здійснюється у файлі __init__.py, де функція *create_app()* ініціалізує *Flask*-додаток, підключає базу даних SQLite та реєструє необхідні *Blueprints (views, auth).*

Модель користувача визначена із бази даних у файлі models.py в класі *User*, який наслідує *UserMixin* для сумісності з *Flask-Login*. Кожен користувач може мати кілька проєктів, що реалізовано через зв’язок "один до багатьох" між класами *User* та *Project* за допомогою SQLAlchemy (рис. 3.2).

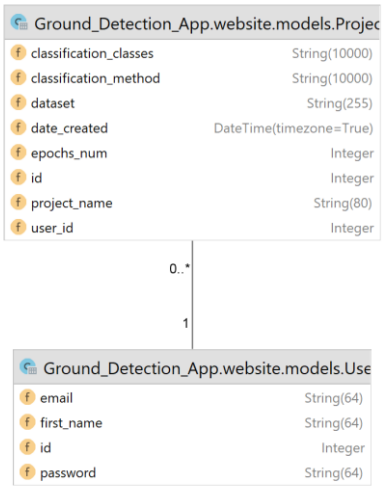


Рисунок 3.2 – Діаграма зв’язків бази даних

Основна взаємодія backend та frontend відбувається у файлі views.py. Цей файл призначено для обробки HTTP-запитів, пов’язаних з управлінням проєктами користувачів, завантаженням датасетів, запуском обраних методів класифікації, побудовою аналітичних графіків (розподіл класів, спектральні зразки, огляд даних, результати класифікації), а також для виводу цих результатів на відповідних веб-сторінках (html-шаблонах). Також views.py забезпечує збереження та оновлення проєктів у базі даних,

динамічне відображення візуалізацій, а також запуск шаблонів HTML із результатами аналізу та класифікації зображень.

Розглянемо backend-складову проєкту. При вході на сайт користувач має пройти автентифікацію або реєстрацію, що здійснюються за допомогою інструментів *Flask*, а саме підмодулів *Flask-Login* для керування сесіями, у які користувач використовує акаунт та для хешування паролів для безпечного створення акаунтів.

Функціонал входу у систему прописаний у файлі auth.py, що відповідає за маршрутизацію запитів до сторінок входу (/*login*), виходу (/*logout*) та реєстрації (/*sign-up*) (рис. 3.3).

```
Sofiia_Halapova
@auth.route( rule: '/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':...
    return render_template( template_name_or_list: "login.html", user=current_user)


Sofiia_Halapova
@auth.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('auth.login'))


Sofiia_Halapova
@auth.route( rule: '/sign-up', methods=['GET', 'POST'])
def sign_up():
    if request.method == "POST":...
    return render_template( template_name_or_list: "sign_up.html", user=current_user)

```

Рисунок 3.3 – Приклад функцій *login()*, *logout()* та *sign_up()*

Під час входу система перевіряє, чи існує користувач з таким email у базі даних. Якщо користувача знайдено, введений пароль перевіряється шляхом порівняння з хешем, збереженим у базі даних. У разі успіху викликається функція login_user, яка створює сесію для користувача. Усі повідомлення про помилки або успіх передаються через функцію flash.

Реєстрація включає валідацію вхідних даних: перевірку довжини email та імені, збіг паролів, довжину паролю тощо. У разі успішної валідації

створюється новий об'єкт *User*, пароль хешується за допомогою *generate_password_hash*, і дані зберігаються в базі даних через SQLAlchemy.

Після успішного входу, реєстрації чи виходу, застосунок візуалізує html-шаблони (home.html; login.html, якщо користувач вийшов із облікового запису або виникла помилка при вході; sign_up.html, якщо виникла помилка при реєстрації) із переданими даними із бази даних та backend.

Реалізація особистих кабінетів та автентифікації забезпечує захищене зберігання облікових даних, доступ до персонального кабінету та зв’язок користувачів зі створеними ними гіперспектральними проєктами.

Перейдемо до структури самої системи аналізу та класифікації. Даний проєкт реалізовано з модульною структурою, що базується на принципах об’єктно-орієнтованого програмування (ООП) (Додаток А. рис. А.2). При застосуванні ООП, застосунок сприймає все як об’єкт та зосереджений на операціях з об’єктами. Структура ООП базується на класах, успадкуванні, поліморфізму та інкапсуляції.

Розглянемо спочатку клас для аналізу спектрів зображень – *DataEvaluation.* Даний клас візуалізує спектральні характеристики зображень, будує графіки розподілу класів та оцінює результати класифікації. Тобто цей клас будує всі потрібні графіки та діаграми, які пізніше передаються на frontend і викликаються на html-шаблоні через кнопки: *Plot Class Distribution plot; Plot Sample Spectra; Comprehensive review plot.*

У клас передається список класів на зображенні, після чого дані завантажуються методом *load_data*, який використовує допоміжний клас *LoadDataset* для отримання набору даних.

LoadDataset приймає шлях до переданого користувачем файлу або посилання та розархівовує і завантажує дані та їхні правдиві мітки для подальшої обробки. Він також відповідає за виділення із даних довжин хвиль.

Один із ключових інструментів класу – це метод *create_false_color(self, bands=None),* який формує інакші, хибні кольори для зображення, поєднуючи три спектральні канали для створення візуально зрозумілої RGB-картинки. Це особливо корисно для першого візуального огляду гіперспектрального зображення.

Наступний метод *plot_class_distribution(self, save_path)* будує графік розподілу пікселів за класами, показуючи кількісну присутність кожного матеріалу на зображенні, що допомагає оцінити збалансованість датасету.

Метод *plot_sample_spectra(self, save_path, selected_classes=None)* дозволяє порівняти спектральні профілі різних класів — для кожного вибраного класу обчислюється середній спектр на основі всіх пікселів, що йому відповідають. Це надає можливість вивчити, наскільки відрізняються матеріали у спектральному вимірі.

Наступний метод, *plot_comprehensive_review(self, save_path)*, створює загальний огляд зображення: в одному вікні відображаються хибно-кольорове зображення, істинні класи, окремий спектральний канал, а також спектральна дисперсія, яка показує як змінюється спектр в кожному пікселі.

Усі графіки створені вище наведеними функціями зберігаються у вигляді зображень методом *save_plot_to_file(plot_type, save_dir),* який генерує унікальні імена файлів.

У застосунку також створений абстрактний базовий клас *BaseClassifier*, який інкапсулює один функціонал для імплементації усіх моделей машинного навчання. Тобто *BaseClassifier* слугує універсальним інтерфейсом для підготовки гіперспектральних даних до входу в модель.

У даному класі завантажується набір даних гіперспектральних зображень із мітками класів, довжинами хвиль через клас *LoadDataset*.

Клас базового класифікатора для усіх моделей має метод *preprocess_data(pca_components=None, test_size=0.3, random_state=42, scale=True)*, який реалізовує попередню обробку даних для подальшого подання у модель. До задач попередньої обробки даних входять:

нормалізація даних, а саме перетворення 16-бітних значень у діапазон від 0, до 1; конвертація 3D-матриці зображення [довжина × ширина × спектри] до 2D-матриці [пікселі × спектри]; фільтрація фонових пікселів (з міткою 0); розбиття даних на навчальну та тестувальну вибірки; масштабування за допомогою *StandardScaler*; та в залежності від методу може бути потреба знизити розмірність даних, тоді застосовується описаний у 2 розділі алгоритм PCA.

Для подальшого використання функцій *BaseClassifier* класами-нащадками існує абстрактний метод *compute_classification_results()*. Даний метод має бути обов'язково реалізований підкласами, якщо нащадок не виконує цей метод, буде викинуто помилку *NotImplementedError("Must be implemented by subclasses")*.

Розглянемо тепер підкласи *BaseClassifier* зокрема: *GeneratePredictionKNearest, GeneratePredictionLogisticRegression, GeneratePredictionRandomForest* та *GeneratePredictionXGBoost*.

Завдяки успадкуванню базового класифікатора дані класи наслідують набір методів, що попередньо обробляє дані та вимагає імплементації лише основної логіки обраного алгоритму машинного навчання. Базовий клас інкапсулює повторювану логіку: підготовку даних, масштабування, поділ на навчальні й тестові підвибірки, а також збереження результатів класифікації.

Кожен підклас реалізує свій метод *compute_classification_results*, у якому викликається метод *preprocess_data*, успадкований від *BaseClassifier*. Отримані дані із цього методу передаються на вхід у відповідну модель;

- Клас *GeneratePredictionKNearest* класифікує методом k-найближчих сусідів, де значення k встановлюється відповідно до кількості класів;
- Клас *GeneratePredictionLogisticRegression* використовує логістичну регресію з фіксованим обмеженням на кількість ітерацій;

- У *GeneratePredictionRandomForest* застосовується ансамблевий метод випадкового лісу з 100 заданими деревами;
- *GeneratePredictionXGBoost* реалізує алгоритм XGBoost, у якому додатково використовується пониження розмірності через PCA, що дозволяє покращити продуктивність моделі і зменшити перенавчання.

Після обробки даних моделлю викликається метод із *BaseClassifier* - *flattent_data*, який повертає передбачену класифікацію у вигляді графіку розпізнаних класів.

Останній спосіб класифікувати об’єкти на зображеннях – це CNN (згорткова нейронна мережа).

Клас *GeneratePredictionCNN* реалізовує повноцінний алгоритм системи класифікації об’єктів на HSI, цей клас теж успадковує *BaseClassifier*.

У конструкторі задаються та ініціалізуються параметри: *class_names* (список назв класів на зображеннях), *epochs* (кількість епох для навчання CNN), *numComponents* (к-сть компонентів для зменшення розмірностей алгоритмом PCA), *patch_size* (розмір відступу навколо кожного пікселя, для формування вхідного фрагмента даних, який буде поданий до CNN для класифікації) та *test_size* (розмір набору даних для тестування виділеного із повного датасету).

Першочергово дані зменшуються алгоритмом PCA, описаним у розділі вище.

Фрагменти пікселів (*patches*) будуються розміром 7x7 навколо кожного пікселя, фрагменти з нульовими значеннями ігноруються.

Далі проводиться аугментація даних, а саме підвищення стійкості моделі до змін. Даний процес потрібен для штучного розширення тренувального набору даних шляхом видозмінення існуючих тренувальних зразків. Для задачі аугментації даних застосовуються випадкові зміни, зокрема горизонтальне та вертикальне віддзеркалення пікселів, а також обертання на випадкові кути, що дозволяє моделі краще розпізнавати

об’єкти незалежно від їх положення чи орієнтації. Такий підхід зменшує ризик перенавчання, покращує узагальнення моделі та підвищує її стійкість до шумів, штучних мінімумів та максимумів у вхідних даних.

Після того, як дані нормалізовані, розбиті на патчі та «зміцнені» аугментацією, задається та починає навчатись модель.

У функції *train_model(self, X_train, y_train)* відбувається створення та навчання CNN, яка складається з кількох основних шарів (рис. 3.4).

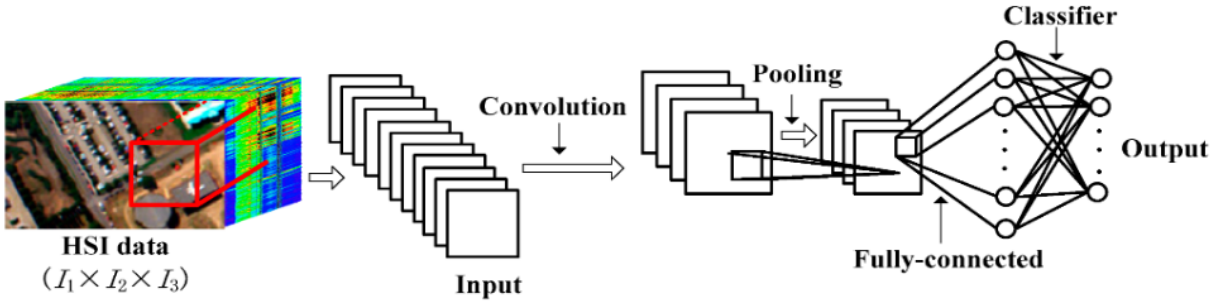


Рисунок 3.4 – Шари згорткової нейронної мережі

Conv2D(32, (3, 3), activation='relu') – це перший згортковий шар, який застосовує 32 фільтри розміром 3×3 для виявлення певних характерних ознак пікселів (контури, текстура тощо) у кожному виділеному раніше фрагменті зображення.

Наступний шар – *MaxPooling2D((2, 2))*, який зменшує просторові розміри ознак (після згортки) шляхом вибору максимального значення в кожному вікні 2×2. Шар *MaxPooling* допомагає зменшити обчислювальну складність і запобігає перенавчанню моделі.

Наступним виступає шар *Flatten()*, який перетворює трьохвимірний тензор ознак у вектор, необхідний для подальшої обробки у щільних шарах.

Шар *Dense(128, activation='relu')* є повнозв’язним прихованим шаром з 128 нейронами, який навчений виявляти складні взаємозв’язки між ознаками. Цей прошарок з'єднує кожен нейрон попереднього з кожним нейроном даного шару. Даний шар отримує вхідний вектор чисел із зазначеними вагами із шару *Flatten*, де ваги визначають наскільки сильно якась ознака впливає на класифікацію. Шар *Dense* додає до цих даних змінну,

константу (bias) для зсуву активації. У кінці шар застосовує функцію активації, на цьому етапі це *ReLU (Rectified Linear Unit)*. *Dense* використовує *ReLU* (18), щоб модель навчилась складним залежностям між даними. Дана лінійна функція повертає вхідне значення без змін, якщо воно позитивне (більше 0) або нуль, якщо значення негативне (менше 0).

$$f(x) = max(0, x) \tag{18}$$

де:

x – вхідний параметр у нейрон

Останній шар – *Dense(y_train.shape[1], activation='softmax')* – це вихідний класифікаційний шар, у якому кількість нейронів дорівнює кількості заданих користувачем класів. Даний шар використовує функцію активації *softmax*, що була розглянута у розділі аналізу логістичної регресії.

Тобто дана ф-ція перетворює вихід на ймовірності належності кожного фрагменту (пату) до певного класу.

Далі модель визначає як саме нейронна мережа буде навчатись. Як функція втрат використовується *categorical_crossentropy* (для багатокласової класифікації) і оптимізатором *Adam*.

Ф-ція втрат вимірює наскільки передбачення моделі відрізняються від правдивих міток. Чим менший показник функції втрат – тим краще модель класифікує.

Оптимізатор у нейронній мережі потрібен для визначення як ваги та зміщення (bias) коригуються під час навчання для мінімізації ф-ції втрат. Для навчання конкретної CNN було обрано оптимізатор *Adam*, оскільки він автоматично налаштовує темп навчання (*learning rate*) під кожну вагу. Даний аспект робить *Adam* дуже надійним.

Після цього модель навчається на тренувальних даних (*X_train, y_train*) протягом заданої користувачем кількості епох.

Після отримання результатів будь-яким з наведених методів, у views.py відбувається передача згенерованих графіків класифікації або аналізу на frontend-складову проєкту.

Метод класифікації обирається користувачем через веб-інтерфейс під час створення або редагування проєкту, після чого ця інформація зберігається у базі даних, передається на backend, а там у свою чергу запускається обраний алгоритм із описаних вище. При виконанні аналізу система звертається до даних у базі, динамічно ініціалізує відповідний класифікатор, тренує модель на вибраному наборі даних та проводить класифікацію. Після завершення процесу результати, а саме передбачені класи, метрики точності, матриця плутанини та інші описані вище графіки, автоматично генеруються й виводяться на відповідних сторінках застосунку для зручного перегляду користувачем.

3.4 Демонстрація готового веб-ресурсу

Даний підрозділ кваліфікаційної роботи присвячено демонстрації функціональності розробленого веб-застосунку для аналізу та класифікації гіперспектральних зображень.

Нижче продемонстровано як реалізовано взаємодію користувача з системою, ілюструючи реалізацію всіх заявлених можливостей.

Стартова сторінка веб-застосунку призначена для входу зареєстрованих користувачів. Інтерфейс надає поля для введення логіну та паролю, якщо це зареєстрований користувач.

Також застосунок надає можливість створити новий обліковий запис, надавши необхідні дані, що дозволяє їм отримати доступ до особистого кабінету та функціоналу аналізу гіперспектральних зображень.

Після реєстрації або входу у систему, розгортається особистий кабінет користувача, який стає доступним після успішної аутентифікації. Ця сторінка є центральним хабом для управління проєктами. Тут відображаються всі збережені проекти користувача, надаючи опції для їх перегляду, перейменування та видалення. Також є функціонал для створення нового проєкту аналізу та класифікації гіперспектральних зображень (Додаток А, рис. А.3).

Сторінка створення нового проєкту – це вікно, яке надає інтерфейс для вводу назви проєкту, завантаження даних через архів із гіперспектральними даними або посилання з веб-порталу *Kaggle*. Застосунок підтримує такі формати наборів даних як: *.mat* (файли MatLab), *.npy* (файли з масивами NumPy), *.npz* (стиснений архів, який використовується бібліотекою NumPy у Python для зберігання масивів NumPy).

Також при створенні нового проєкту користувач має вказати назви класів та обрати метод для класифікації із спадаючого меню у формі (XGBoost, Random Forest Classifier, Logistic Regression, K-nearest neighbors та CNN). При виборі CNN як класифікатора, можна додатково обрати кількість епох для нейронної мережі (рис. 3.5). Після натискання кнопки “Create project”, дані введені користувачем передано у систему застосунку для подальшої обробки набору даних, а користувача направлено на сторінку нового проєкту.

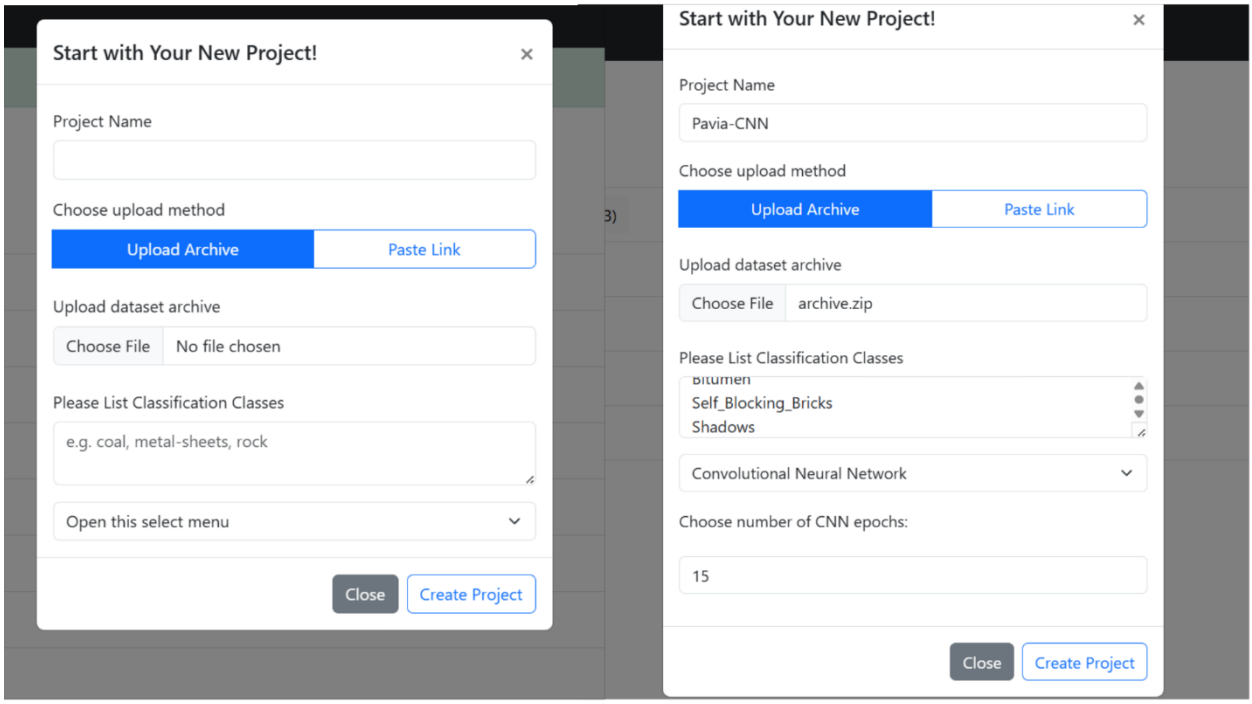


Рисунок 3.5 – Інтерфейс створення нового проєкту

Після створення нового проєкту або вибору існуючого, відкривається HTML-сторінка. Цей робочий простір містить основну інформацію про

проєкт, перелік класів, наявні дані та ключові інструменти для їх обробки (рис. 3.6).

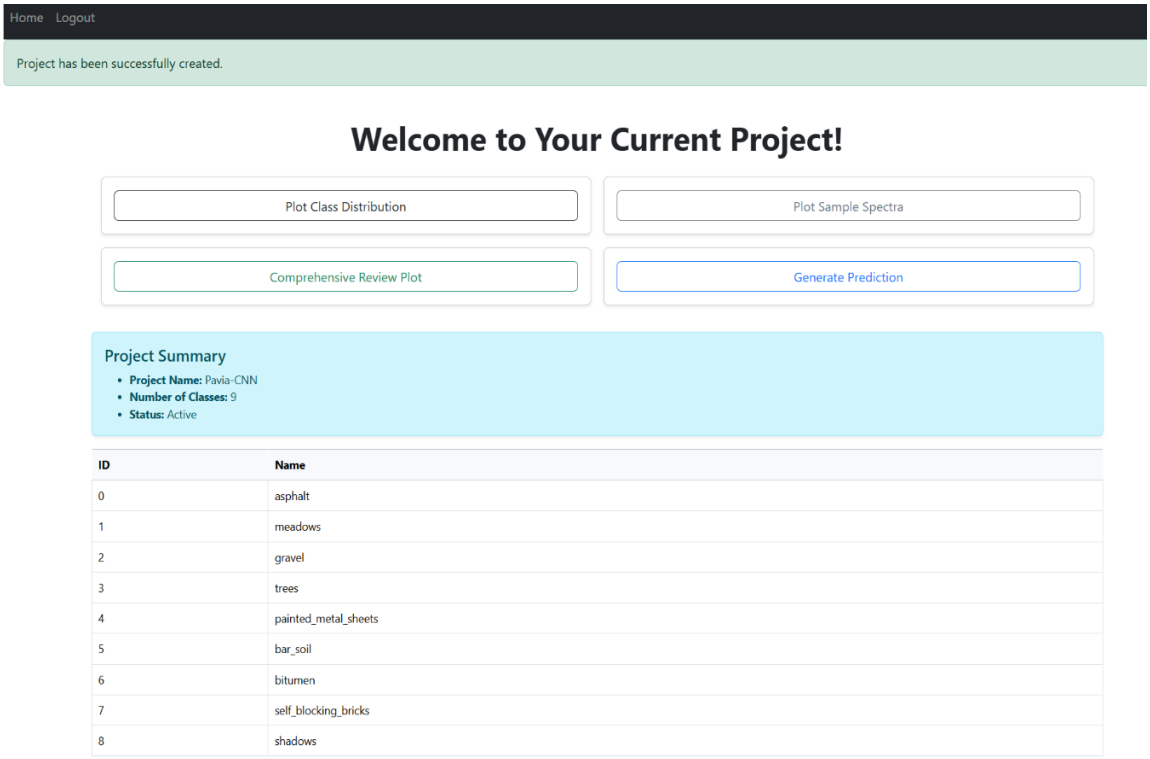


Рисунок 3.6 – Інтерфейс сторінки робочої зони проєкту.

У межах робочого простору проєкту користувачеві доступні чотири інтерактивні кнопки для генерації аналітичних візуалізацій та виконання класифікаці:

- *Class Distribution plot* (графік розподілу класів, що показує частоту або відсоток кожного класу в завантаженому наборі даних. Цей графік (стовпчаста діаграма) допомагає зрозуміти баланс класів у даних.)
- *Sample Spectra Plot* (застосунок дозволяє обирати, спектри яких класів користувач хоче проаналізувати та демонструє лінійні графіки зразкових спектрів відбиття для обраних пікселів класу. Це дозволяє користувачеві візуально оцінити спектральні характеристики різних матеріалів.)
- *Comprehensive review plot* (комплексний огляд обраного гіперспектрального зображення. Даний графік дозволяє отримати повне розуміння характеристик гіперспектрального зображення перед його

аналізом. Графік поділяється на: підграфік псевдо-кольорового зображення на основі вибраних спектральних діапазонів; підграфік ground truth або істинних класів, на якому відображається істинна класифікація кожного пікселя зображення; підграфік окремого спектрального каналу зображення, вибирається канал із середнім значенням з усього діапазону довжин хвиль; підграфік спектральної дисперсії або стандартного відхилення спектральних значень для кожного пікселя по всіх довжинах хвиль. Висока спектральна дисперсія вказує на те, що спектральна сигнатура пікселя є більш "мінливою" або різноманітною, тоді як низька дисперсія свідчить про більш однорідний спектр.)

- *Класифіковане зображення.*

Усі згенеровані графіки наведені вище можуть бути завантажені на пристрій користувача за допомогою відповідної кнопки, забезпечуючи можливість збереження та подальшого використання результатів аналізу.

Після натиснення кнопки *«Generate prediction»,* запускається процес класифікації гіперспектрального зображення. Після завершення класифікації, застосунок автоматично перенаправляє користувача на нову сторінку, де виводяться два ключові зображення: зображення з класифікованими класами (зліва), що показує передбачені класи для кожного пікселя, та істинні зображення (справа) для візуального порівняння (Додаток А, рис. А.4).

Для класифікації за допомогою Convolutional Neural Network (CNN), на цій сторінці присутня додаткова кнопка, яка дозволяє змінити кількість епох навчання та повторно запустити класифікацію (Додаток А, рис. А.5).

Також на сторінці результатів класифікації можна отримати модальне вікно, яке з'являється після натискання відповідної кнопки. У вікні відображається матриця невідповідностей (*Confusion Matrix*), що надає детальну метрику точності класифікації для кожного класу, показуючи кількість правильно та неправильно класифікованих пікселів (рис. 3.7).

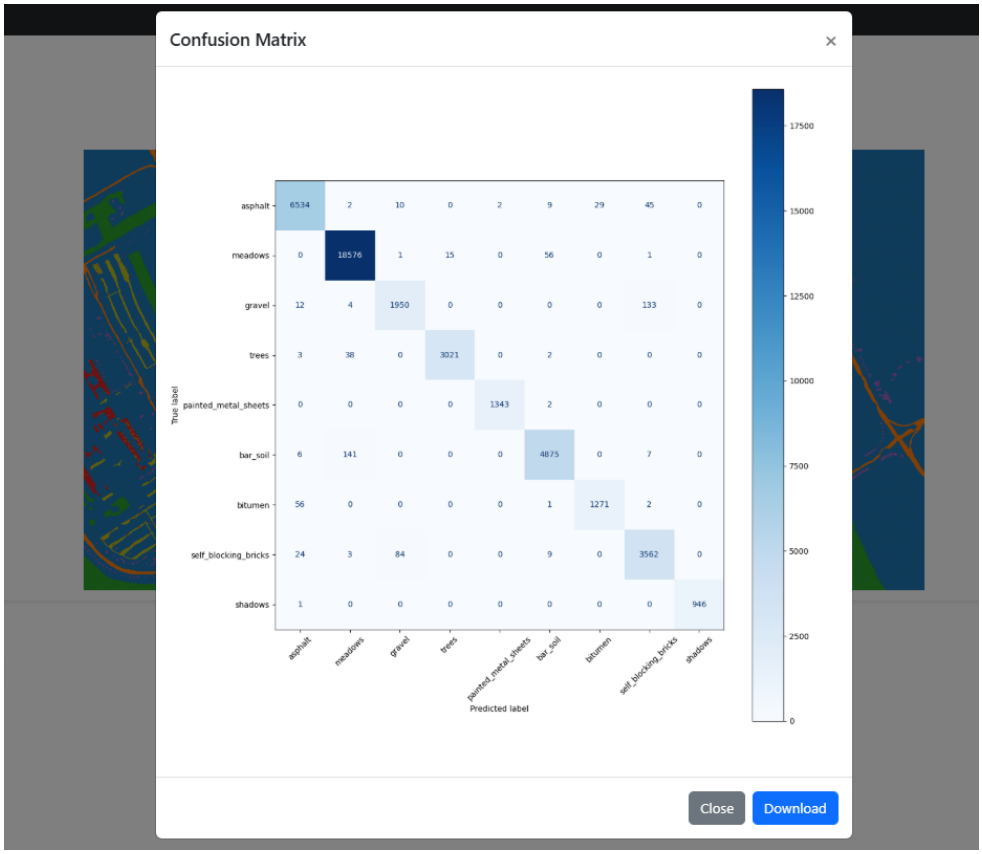
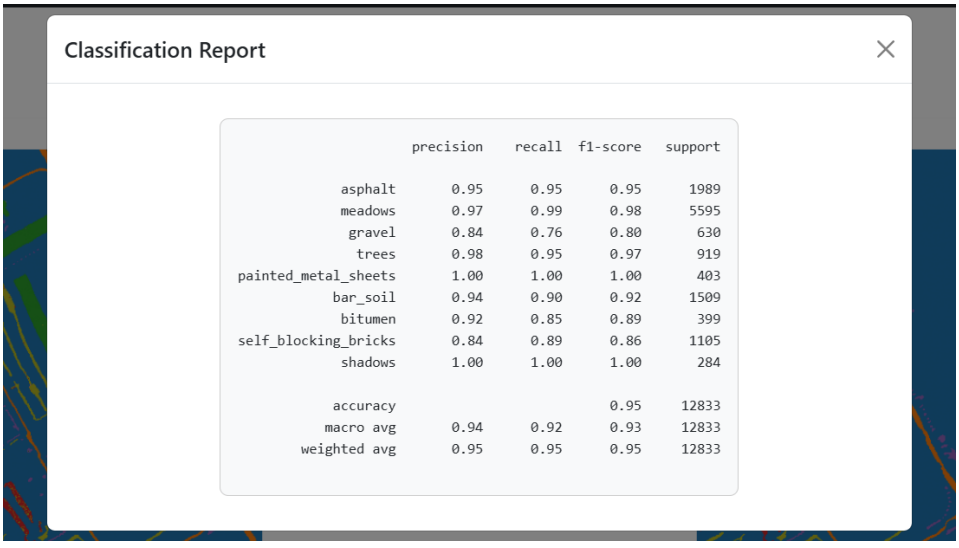


Рисунок 3.7 – Згенерована та виведена у вікно матриця невідповідностей передбачення моделі.

Також застосунок дозволяє згенерувати звіт класифікації (Classification Report), який надає такі метрики, як точність (precision), повнота (recall) та F1-оцінка для кожного класу, а також загальну точність моделі (рис. 3.8).



	precision	recall	f1-score	support
asphalt	0.95	0.95	0.95	1989
meadows	0.97	0.99	0.98	5595
gravel	0.84	0.76	0.80	630
trees	0.98	0.95	0.97	919
painted_metal_sheets	1.00	1.00	1.00	403
bar_soil	0.94	0.90	0.92	1509
bitumen	0.92	0.85	0.89	399
self_blocking_bricks	0.84	0.89	0.86	1105
shadows	1.00	1.00	1.00	284
accuracy			0.95	12833
macro avg	0.94	0.92	0.93	12833
weighted avg	0.95	0.95	0.95	12833

Рисунок 3.8 – Згенерований та виведений у вікно звіт класифікації передбачення моделі.

3.5 Аналіз наборів даних для тестування моделей класифікації

Для задачі класифікації було використано вище описані алгоритми машинного навчання: Random Forest Classifier, XGBoost, Logistic Regression, K-nearest neighbors, CNN.

Кожна модель показала свої особисті результати та точність у різних аспектах класифікації.

Моделі були протестовані на трьох різних наборах даних: Indian Pines, округ Індіан-Пайнс, штат Індіана, США (сенсор *AVIRIS*); Salinas, Салінас-Веллі, Каліфорнія, США (сенсор *AVIRIS*) та Pavia University, територія університету у місті Павія, Італія (сенсор *ROSIS*).

3.5.1 Сенсор AVIRIS

Розглянемо детально як було зібрано набори даних для тестування. Indian Pines та Salinas було знято за допомогою сенсору *AVIRIS (Airbone Visible/Infrared Imaging Spectrometer),* авіаційного спектрометра для задач зондування земної поверхні, що розпізнає видимий та інфрачервоний діапазони хвиль. Даний сенсор має високу спектральну роздільну здатність, внаслідок чого дозволяє отримувати спектральні дані та канали для кожного пікселя на знятому зображенні. Отримання надзвичайно детальної інформації про поверхню землі та пікселі зумовлено тим, що сенсор робить калібровані фото спектральної яскравості у 224 спектральних безперервних каналах із довжиною хвиль від 400 до 2500 нанометрів.

Даний сенсор робить знімки завдяки тому, що його прикріплюють на літальні апарати, наприклад такі як: реактивний літак NASA ER-2, що літає на висоті 20 км над рівнем моря зі швидкістю 730 км/год; турбогвинтовий літак Twin Otter International, що літає на висоті 4 км над землею зі швидкістю 130 км/год; Scaled Composites' Proteus, висотний літак з тандемним крилом, розроблений для дослідження можливості використання літаків як висотних телекомунікаційних станцій та NASA WB-57, висотна

дослідницька повітряна платформа, що піднімається до висоти 18 км над рівнем землі та використовується для наукових досліджень як от збір гіперспектральних даних.

3.5.2 Набір даних Indian Pines

Перейдемо до розгляду датасет Indian Pines, який містить зображення розміром 145×145 пікселів із 224-ма спектральними каналами. Набір даних містить 16 класів, тобто різних типів покриття, як от: кукурудза, соя, ліс, пшениця, овес тощо.

Даний датасет є більш складним для класифікації порівняно з іншими двома, шо будуть розглянуті пізніше. Гірша класифікація відбувається через те, що у Indian Pines багато класів належать до схожих сільськогосподарських культур (наприклад, різні типи сої чи кукурудзи). Такі класи мають дуже близькі спектральні характеристики, що ускладнює розрізнення між ними, навіть для моделі згорткової нейронної мережі. Також цей набір даних незрівноважений більше ніж інші датасети, оскільки деякі класи мають значно менше пікселів, ніж інші, тому моделі погано навчаються на малочисельних класах. Також розмір зображень Indian Pines – лише 145×145 пікселів, тому для деяких моделей (особливо глибоких, як CNN) цього може бути недостатньо для повноцінного навчання.

3.5.3 Набір даних Salinas

Наступний набір даних знятий сенсором *AVIRIS* – Salinas. Цей датасет містить зображення розміром 512×217 пікселів та 224 спектральних каналів. На зображеннях міститься 16 класів, тобто різних видів сільськогосподарських культур.

Особливість даного набору у тому, що він має вищу роздільну здатність і чіткіші межі між класами порівняно з Indian Pines,що робить його зручнішим для тестування алгоритмів машинного навчання.

3.5.4 Сенсор ROSIS

Набір даних Pavia University було зібрано за допомогою сенсору *ROSIS (Reflective Optics System Imaging Spectrometer).* Даний сенсор це компактний програмований спектрометр зображень, розроблений для детального дистанційного зондування з повітряних та потенційно космічних платформ. Його ключовою особливістю є використання повністю відбивної оптики та матричних CCD-детекторів.

CCD-детектор – це аналогова мікросхема, що перетворює світлові сигнали в електричні заряди, що дозволяє сформувати пікселі зображення. Коли фотони потрапляють на світлочутливий елемент CCD, вони вибивають електрони, створюючи електричний заряд. CCD-детектори відомі своєю високою чутливістю до світла, що дозволяє їм отримувати якісні зображення навіть в умовах низької освітленості.

Принцип роботи *ROSIS* полягає у тому, що коли літак летить, оптична система *ROSIS* збирає світло з вузької смуги на поверхні Землі. Це світло потім розкладається дифракційною ґраткою на сотні вузьких спектральних каналів (від видимого до ближнього інфрачервоного діапазону). CCD-детектор одночасно реєструє інтенсивність світла в кожному з цих каналів для кожного пікселя вздовж сканованої лінії. По мірі руху літака, сенсор безперервно знімає нові лінії, формуючи детальне гіперспектральне зображення.

3.5.5 Набір даних Pavia University

Датасет Pavia University знятий сенсором *ROSIS*. Розмір зображень даного набору 610×340 пікселів та кількість спектральних каналів – 115. Кожне зображення містить по 9 класів (будівлі, дороги, дерева, ґрунт, тіні, трава тощо). Кожен піксель зображення відповідає площі 1.3 метра на 1.3 метра на поверхні землі.

Зображення були отримані під час польотної кампанії над містом Павія, Північна Італія. Територія, яку охоплює зображення, включає кампус Університету Павії та прилеглі міські об'єкти.

На відміну від Indian Pines та Salinas, які в свою чергу охоплюють сільськогосподарські угіддя зі значною спектральною схожістю між класами, Pavia University зображує околиці міста. Об'єкти у міській забудові (будівлі, дороги, тіні) часто мають більш виразну спектральну роздільну здатність та чіткіші просторові межі, що може полегшувати завдання класифікації порівняно з Indian Pines, де межі менш чіткі.

3.6 Порівняння якості класифікації різними моделями

У ході тестування застосунку було зібрано дані для оцінки точності та якості класифікації різними, згаданими вище, методами машинного навчання (табл. 3.1).

Таблиця 3.1 Порівняння значень метрик різних методів машинного навчання для задачі класифікації гіперспектральних зображень

Метод	***Accuracy***	***Precision (Macro)***	***Recall (Macro)***	***F1-score (Macro)***	***Precision (Weighted)***	***Recall (Weighted)***	***F1-score (Weighted)***
XGBoost	0.96	0.98	0.98	0.98	0.96	0.96	0.96
Log Reg	0.93	0.97	0.96	0.97	0.93	0.93	0.93
Random Forest	0.95	0.98	0.97	0.97	0.95	0.95	0.95
KNN	0.87	0.92	0.92	0.92	0.87	0.87	0.86
CNN	0.99	0.99	0.99	0.99	0.99	0.99	0.99

Було проаналізовано такі метрики у варіантах macro (спосіб підрахунку метрик класифікації, коли спочатку обчислюються показники для кожного класу окремо, а потім знаходиться середнє арифметичне цих показників без урахування ваги кожного класу) та weighted (спосіб

підрахунку метрик класифікації, коли метрики обчислюються окремо для кожного класу, а потім усереднюються з урахуванням кількості зразків у кожному класі) як:

- *Accuracy* (точність, тобто правильно класифіковані класи серед усіх значень вибірки);
- *Precision* (точність позитивного прогнозу, тобто частка правильно передбачених позитивних випадків серед усіх, які модель класифікувала як позитивні);
- *Recall* (повнота передбачення, тобто частка правильно передбачених позитивних випадків серед усіх дійсно позитивних.);
- *F1-score* (середнє значення precision і recall, що враховує обидва показники одночасно).

Перейдемо до розгляду результатів моделей. Згорткова нейронна мережа показала найкращі результати за всіма метриками – 99% точності, *precision, recall* та *F1-score* як у *macro*, так і у *weighted* варіантах. Це свідчить про здатність CNN ефективно класифікувати дані як для часто представлених, так і для рідших класів, що особливо важливо при роботі з гіперспектральними зображеннями, де просторові та спектральні залежності відіграють ключову роль. Глибока архітектура CNN дозволяє виявляти складні, нелінійні зразки в зображеннях, що й забезпечує її перевагу над іншими моделями.

XGBoost і Random Forest також продемонстрували високі результати (від 95% до 98% за всіма метриками), однак XGBoost, попри найвищий macro precision (98%), має дещо нижчі показники *weighted recall* та *weighted F1-score* – 96%, що може свідчити про вищу кількість помилок у класифікації класів із меншою кількістю зразків. Це звично для моделей бустингу, які можуть бути чутливими до перекосів у даних або особливостей попередньої обробки даних.

У свою чергу, Random Forest показав стабільні результати і трохи кращу збалансованість між метриками, що робить його надійним вибором у випадках, коли важлива стійкість до шуму.

Логістична регресія поступається вищеописаним моделям, демонструючи гарні, але нижчі показники (93–97%). Така поведінка зумовлено обмеженою здатністю даного алгоритму моделювати складні залежності у даних. Логістична регресія не враховує просторову структуру пікселів у зображеннях, її ефективність знижується у задачах класифікації зображень без попереднього виділення ознак.

Найгірші результати показав KNN — 87% точності і найнижчі значення *precision, recall* і *F1-score* (усі на рівні (86%–92%). Метод найближчих сусідів не навчається, а просто шукає найближчі приклади, що робить його чутливим до високої розмірності та шуму в даних. Крім того, через незбалансованість класів та високу кількість спектральних ознак, метрика відстані, на якій базується KNN, втрачає свою ефективність.

ВИСНОВКИ

У результаті виконання даної кваліфікаційної роботи було розроблено інтерактивний веб-застосунок для аналізу спектрів HSI та класифікації об’єктів на гіперспектральних зображеннях із використанням методів машинного навчання.

Розроблений веб-ресурс надає зручний інтерфейс для користувача, який дозволяє завантажити власний HSI-датасет, обрати відповідний метод класифікації (CNN, XGBoost, Random Forest, Logistic Regression, KNN), налаштувати параметри моделі та отримати класифіковане зображення разом з метриками якості класифікації.

Проаналізовано та застосовано фреймворк Flask, впровадження якого дозволило з легкістю розробити веб-частину застосунку, розгорнувши ресурс на локальному порті *http://127.0.0.1:5000*, а інтеграція бібліотек машинного навчання (Keras, scikit-learn, XGBoost тощо) дала змогу реалізувати та порівняти існуючі алгоритми для задачі класифікації.

Визначено унікальність розробленої системи, а саме її орієнтація на обробку багатоканальних гіперспектральних зображень, що значно відрізняє її від усіх існуючих рішень, зосереджених переважно аналізі, розпізнаванні та передбаченнях RGB-зображень.

Проаналізовано результати, що демонструє система. Було отримано гарні результати класифікації завдяки ретельній попередній обробці зображень, використанню методів зменшення розмірності та аугментації даних.

Дана робота є потрібною та актуальною, оскільки розроблений веб-застосунок можна використовувати у прикладних задачах аграрного моніторингу, екології, медицини та оборони, зокрема в умовах актуальних потреб України щодо виявлення вибухонебезпечних предметів, а саме розмінування територій бойових дій.

ПЕРЕЛІК ПОСИЛАНЬ

1. Harris G. Image Noise: Why It Occurs And How to Avoid It. Learning With Experts. URL: https://www.learningwithexperts.com/blogs/articles/image-noise-why-it-occurs-and-how-to-avoid-it?srsltid=AfmBOooulX6qIHOIY08GKxLFAW0J-ISGVvbugljL8k0bsuIXld8rY1EV. Дата звернення: 03.04.2025.
2. Zakaria J. Data Standardization: How to Do It and Why It Matters. BuiltIn. URL: https://builtin.com/data-science/when-and-why-standardize-your-data. Дата звернення: 18.04.2025.
3. Xiaochuan Y., Mary B. Ozdemir, M. K. Joshie. An Overview of Hyperspectral Image Classification by Data-driven Deep Learning. Frontiers in Computing and Intelligent Systems. Дата публікації: 2023. Т. 5. С. 107–110.
4. Zhifei Chen, Yang Hao, Qichao Liu, Yuyong Liu, Mingyang Zhu, Liang Xiao. Deep Learning for Hyperspectral Image Classification: A Critical Evaluation via Mutation Testing /. International Journal of Remote Sensing. Дата публікації: 2024.
5. Ekrem Tarık Karan, Zümray Dokur, Tamer Ölmez. Comparison of Small-Sized Deep Neural Network Models for Hyperspectral Image Classification. Computational Intelligence, Data Analytics and Applications. Дата публікації: 15.03.2023. Т. 643. С. 244–256.
6. Mayar A. Shafaey, Mohammed A.-M. Salem, Maryam N. Al-Berry, Hala M. Ebied, Elsayed A. El-Dahshan, Mohammed F. Tolba. Hyperspectral Image Classification Using Deep Learning Technique. Proceedings of the International Conference on Artificial Intelligence and Computer Vision (AICV2020). Дата публікації: 24.03.2020. Т. 1153. С. 334–342.
7. Renlong Hang, Zhu Li, Qingshan Liu, Pedram Ghamisi, Shuvra S. Bhattacharyya. Hyperspectral Image Classification with Attention Aided

CNNs, Cornell University - Computer Vision and Pattern Recognition. Дата публікації: 12.06.2020.

8. Danfeng Hong, Zhu Han, Jing Yao, Lianru Gao, Bing Zhang, Antonio Plaza, Jocelyn Chanussot. SpectralFormer: Rethinking Hyperspectral Image Classification with Transformers, Cornell University - Computer Vision and Pattern Recognition. Дата публікації: 20.05.2021.
9. Haokui Zhang, Ying Li, Yenan Jiang, Peng Wang, Qiang Shen, Chunhua Shen. Hyperspectral Classification Based on Lightweight 3-D-CNN With Transfer Learning, Cornell University - Computer Vision and Pattern Recognition. Дата публікації: 07.12.2020.
10. Hao Zhang, Xu Ma, Xianhong Zhao, Gonzalo R. Arce, Compressive spectral image classification using 3D coded convolutional neural network, Cornell University - Image and Video Processing. Дата публікації: 12.08.2021.
11. Zhifei Chen, Yang Hao, Qichao Liu, Yuyong Liu, Mingyang Zhu, Liang Xiao, Deep Learning for Hyperspectral Image Classification: A Critical Evaluation via Mutation Testing, International Journal of Remote Sensing. Дата публікації: 16.12.2024.
12. Mahesh Pal. Deep learning algorithms for hyperspectral remote sensing classifications: an applied review. International Journal of Remote Sensing. Дата публікації: 16.01.2024. С. 451–491.
13. Ava Vali, Sara Comai, Matteo Matteucci. Deep Learning for Land Use and Land Cover Classification Based on Hyperspectral and Multispectral Earth Observation Data: A Review. International Journal of Remote Sensing. Дата публікації: 03.08.2020.
14. Халапова С.В, Кисіль Т.М., MACHINE LEARNING МОВОЮ ПРОГРАМУВАННЯ JAVA / Науково-практична конференція «Проблеми комп'ютерної інженерії». Збірник тез. – К.: ДУТ, 2022, с. 110-113
15. Халапова С.В, Кисіль Т.М., ВИКОРИСТАННЯ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ АВТОМАТИЗАЦІЇ ТА

16. ОПТИМІЗАЦІЇ ПРОЦЕСІВ ХМАРНОГО ОБЧИСЛЕННЯ / III Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті». Збірник тез. – К.: ДУІКТ, 2023, с. 74-76
17. Халапова С.В, Кисіль Т.М., DALL-E: ГЕНЕРАЦІЯ ТА ЦИФРОВА ОБРОБКА КОНТЕНТУ / Науково-практична конференція «Проблеми комп'ютерної інженерії». Збірник тез. – К.: ДУІКТ, 2023, с. 61-64
18. Халапова С.В, Кисіль Т.М., Бондарчук А.П., ГІПЕРСПЕКТРАЛЬНИЙ АНАЛІЗ ДЛЯ РОЗПІЗНАВАННЯ ТА КЛАСИФІКАЦІЇ МАТЕРІАЛІВ ЗЕМНОЇ ПОВЕРХНІ / Науковий журнал - ТЕЛЕКОМУНІКАЦІЙНІ ТА ІНФОРМАЦІЙ НІ ТЕХНОЛОГІЇ, 2025, с. 11-27
19. Халапова С.В, Кисіль Т.М., ПОРІВНЯЛЬНИЙ АНАЛІЗ АЛГОРИТМІВ XGBOOST ТА RANDOM FOREST CLASSIFIER ДЛЯ ЗАДАЧ КЛАСИФІКАЦІЇ БАГАТОСПЕКТРАЛЬНИХ ЗОБРАЖЕНЬ / V Всеукраїнська Науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті». Збірник тез. – К.: ДУІКТ, 2025, с. 150-153

ДОДАТКИ

ДОДАТОК А. Структурні схеми створеної моделі

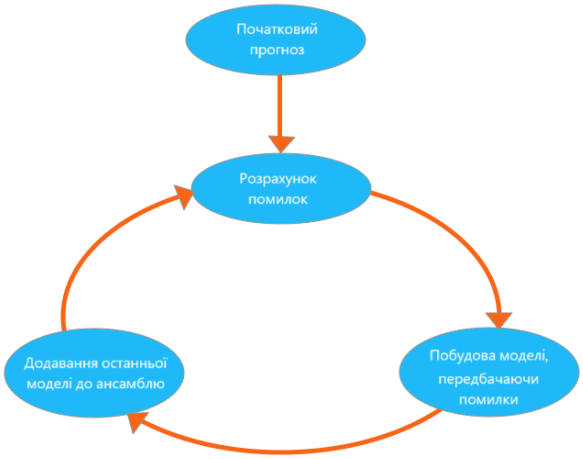


Рисунок А.1 – Схематичне зображення роботи алгоритму XGBoost

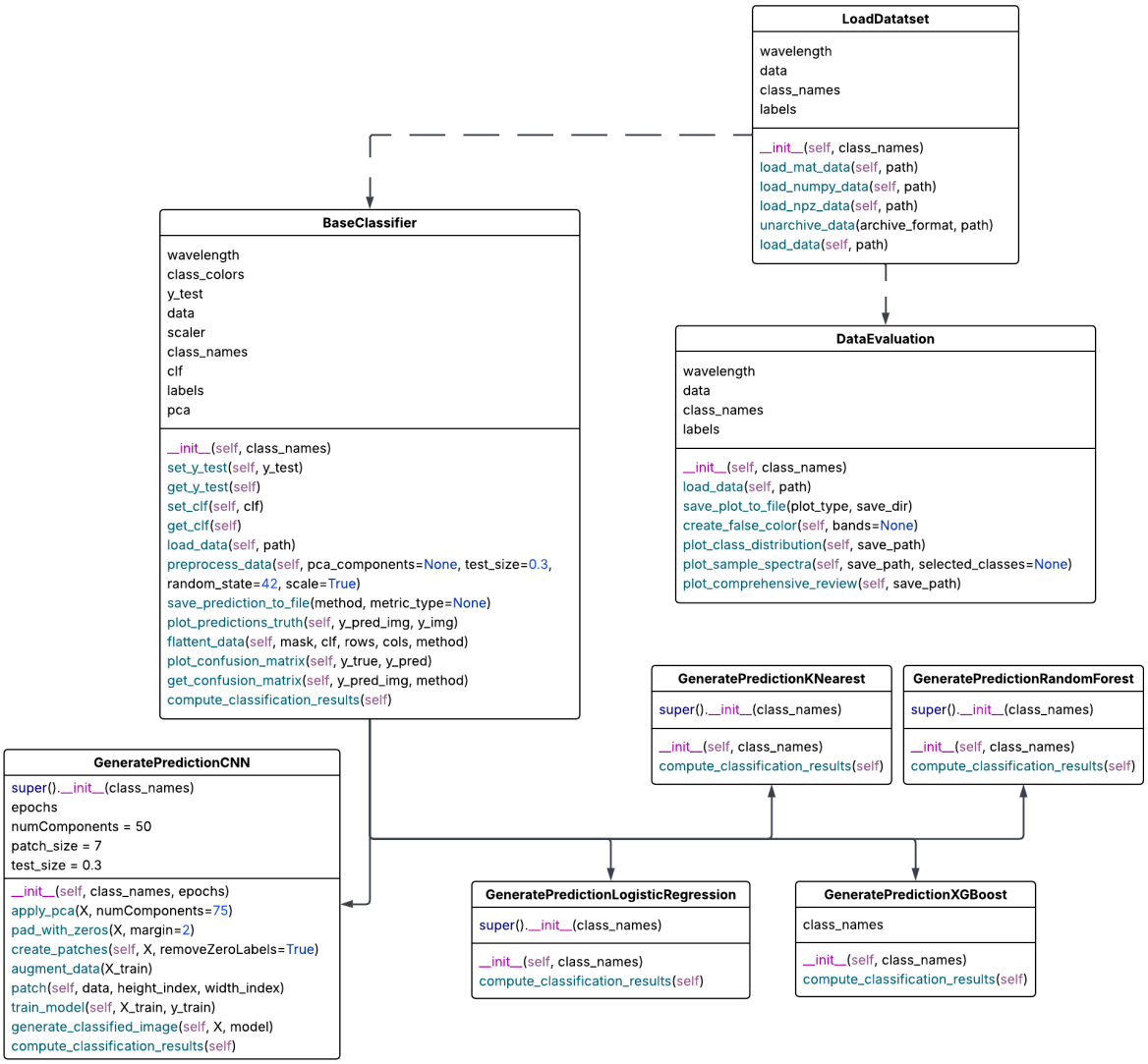


Рисунок А.2 - Класова структура застосунку

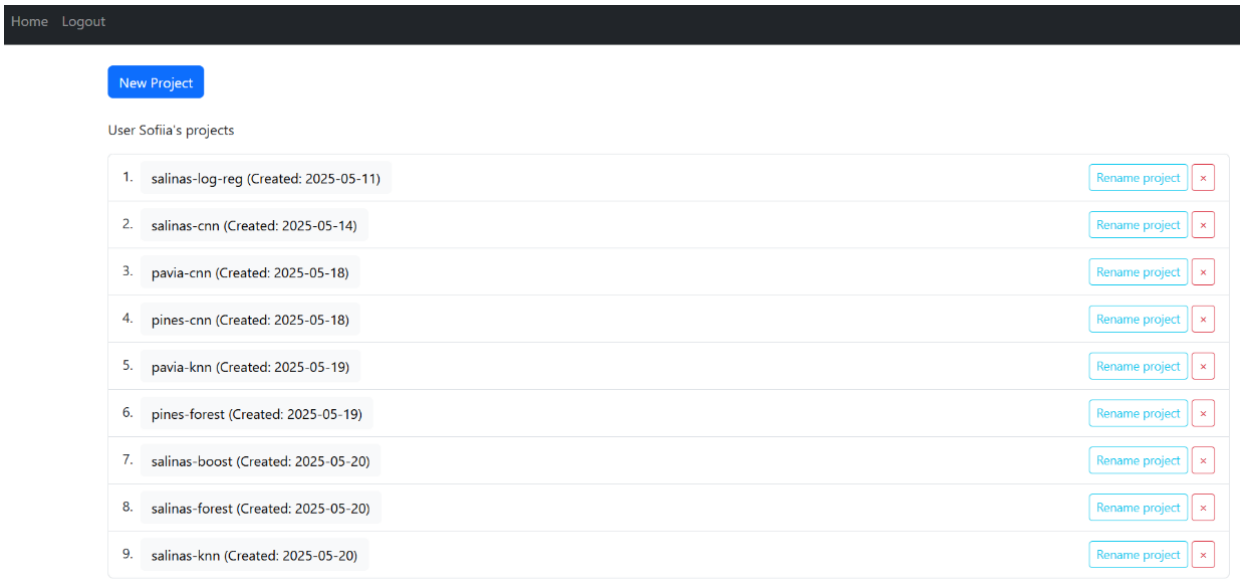


Рисунок А.3 – Особистий кабінет користувача

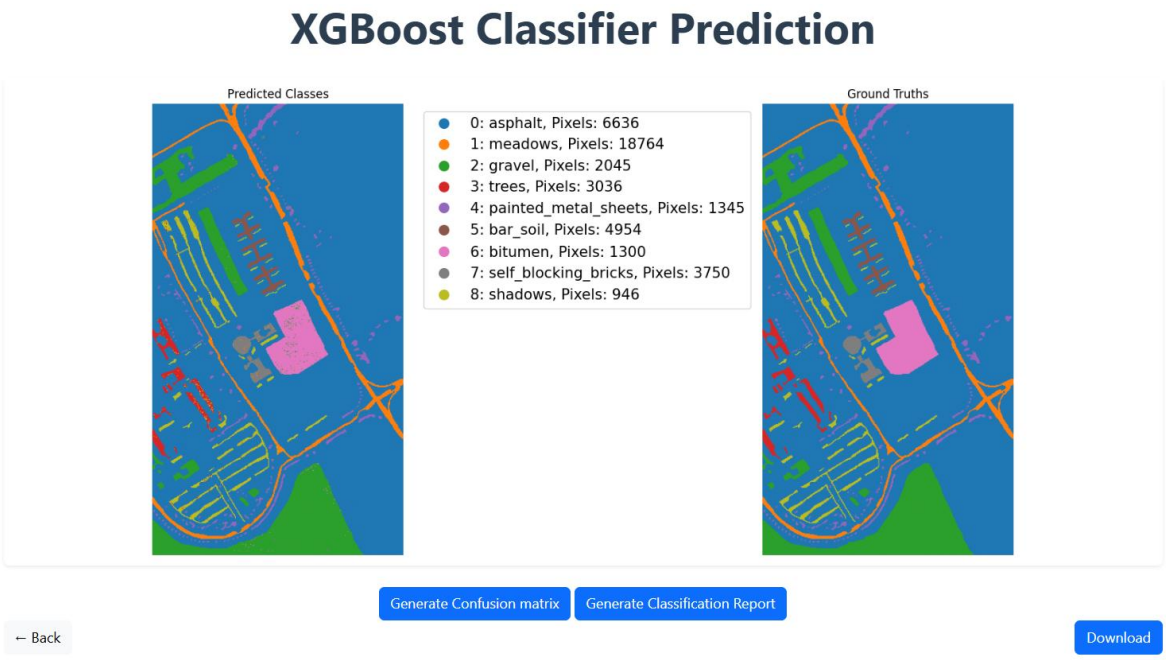


Рисунок А.4 – Інтерфейс виводу класифікованого зображення методом XGBoost.

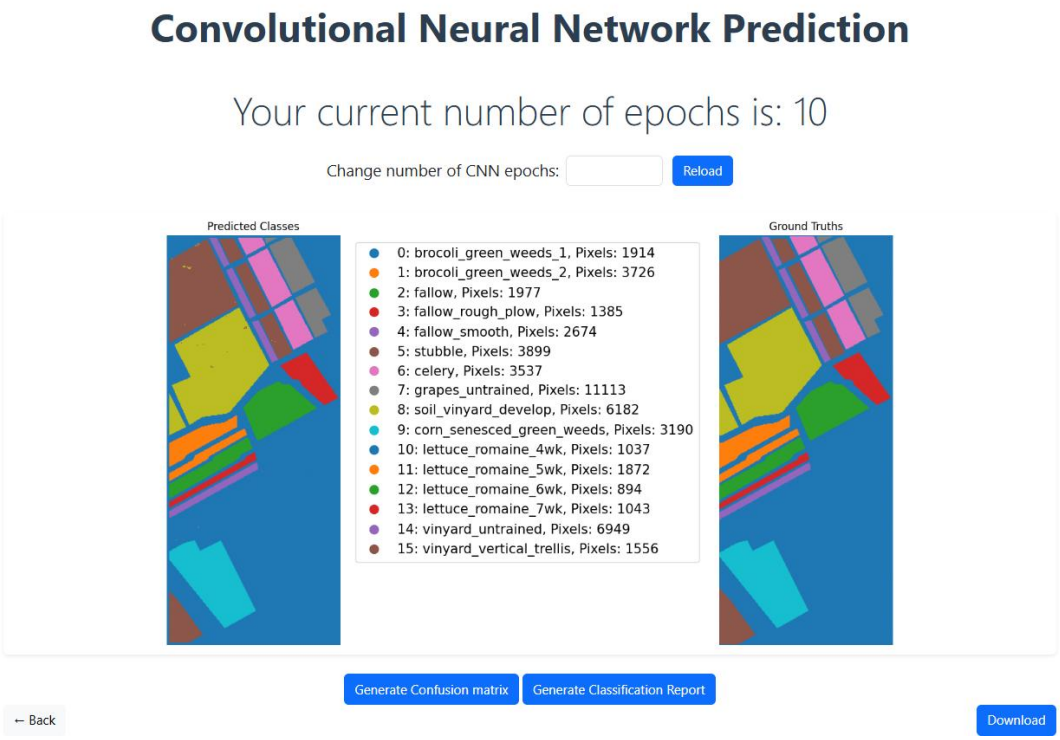


Рисунок А.5 – Інтерфейс сторінки з виведеним класифікованим зображенням CNN.

ДОДАТОК Б. Програмний код розробленого ресурсу

main.py:

```
from website import create_app

app = create_app()

# run flask application (running server)
if __name__ == '__main__':
    app.run(debug=True, port=5000)
    app.config['MAX_CONTENT_LENGTH'] = 100 * 1024 * 1024
```

views.py:

```
import os
import random
import re
import shutil
import string

import numpy as np
from flask import Blueprint, render_template, url_for, redirect, send_file
from flask import request, flash, session, json, jsonify
from flask_login import login_required, current_user
from sklearn.metrics import classification_report
from werkzeug.utils import secure_filename

from .ml_model.data_evaluation import DataEvaluation
from .ml_model.k_nearest_classes_prediction import GeneratePredictionKNearest
from .ml_model.logistic_regression_classes_prediction import
GeneratePredictionLogisticRegression
from .ml_model.neural_network_classes_prediction import GeneratePredictionCNN
from .ml_model.random_forest_classes_prediction import GeneratePredictionRandomForest
```

```
from .ml_model.xg_boost_classes_prediction import GeneratePredictionXGBoost
from .models import Project, db

views = Blueprint('views', __name__)

ALLOWED_EXTENSIONS = {'zip', 'tar', 'gz'}
CURRENT_DIR = os.path.dirname(os.path.abspath(__file__))
STATIC_FOLDER = os.path.join(CURRENT_DIR, 'static')
UPLOAD_DATASETS_FOLDER = os.path.join(STATIC_FOLDER, 'uploads')


def allowed_file(filename, extensions):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in extensions


def get_classification_classes(raw_input: str) -> str:
    cleaned = re.sub(r'[,;|/\\]+', ' ', raw_input)

    cleaned = re.sub(r'\W+', ' ', cleaned)

    words = cleaned.lower().split()
    unique_words = list(dict.fromkeys(words))

    return ' '.join(unique_words)


def load_dataset_file(upload_method, dataset_file, dataset_link, error, project_name):
    kagglehub_pattern = re.compile(
        r"\s*kagglehub\.dataset_download\s*\(\s*\"([a-zA-Z0-9_-]+/[a-zA-Z0-9_-]+)\"\s*\)\s*"
    )
    dataset = None

    if upload_method == "link" and dataset_link:
```

```
        if kagglehub_pattern.match(dataset_link):
            dataset = dataset_link.strip()
        else:
            flash("Invalid dataset link type. Download allowed from kagglehub", category='error')
            error = True
    elif upload_method == "archive" and dataset_file and allowed_file(dataset_file.filename,
ALLOWED_EXTENSIONS):
        filename = secure_filename(dataset_file.filename)
        file_path = os.path.join(UPLOAD_DATASETS_FOLDER, filename)
        # print("views archive file path", file_path)
        dataset_file.save(file_path)
        dataset = unarchive_dataset(file_path, project_name)
    else:
        flash("Invalid dataset file type. Allowed: zip, tar, gz.", category='error')
        error = True

    return dataset, error



def validate_upload_input(project_name, classification_classes,
                          classification_method, dataset_file, dataset_link):
    error = False

    if not project_name:
        flash("Project name field is required.", category='error')
        error = True
    elif Project.query.filter_by(project_name=project_name).first():
        flash("Project with such name already exists!", category='error')
        error = True
    elif not classification_classes:
        flash("Classification classes field is required.", category='error')
        error = True
    elif not classification_method:
```

```
        flash("Classification method field is required.", category='error')
        error = True
    elif not dataset_file and not dataset_link:
        flash("You must upload a dataset file or provide a link.", category='error')
        error = True
    elif dataset_file and dataset_link:
        flash("Please provide either a dataset file or a link, not both.", category='error')
        error = True
    return error


def get_current_project_or_redirect():
    project_id = session.get('current_project_id')
    if not project_id:
        flash("No current project selected.", 'error')
        return None, redirect(url_for("views.home"))

    project = Project.query.get(project_id)
    if not project:
        flash("Project not found.", 'error')
        return None, redirect(url_for("views.home"))

    return project, None


def generate_classes_dictionary(class_names) -> dict:
    class_names = class_names.split()
    class_names_dict = {}
    num = 0
    for i in class_names:
        class_names_dict.update({num: i})
        num += 1
```

```
    return class_names_dict


def render_plot_page(plot_func, template, plot_name):
    project, redirect_response = get_current_project_or_redirect()
    if redirect_response:
        return redirect_response

    class_names_dict = generate_classes_dictionary(project.classification_classes)

    de = DataEvaluation(class_names=class_names_dict)
    de.load_data(project.dataset)

    img_directory = os.path.join(os.path.abspath(STATIC_FOLDER), 'imgs')

    os.makedirs(img_directory, exist_ok=True)

    random_string = "".join(random.choice(string.ascii_letters + string.digits) for _ in range(5))

    full_image_path = os.path.join(img_directory, f'{plot_name}_{random_string}.png')

    image_path = plot_func(de, save_path=full_image_path)

    return render_template(template, image_path=url_for('static', filename=image_path),
user=current_user,
                           project=project)


def unarchive_dataset(file_path, project_name):
    extract_dir = os.path.join(UPLOAD_DATASETS_FOLDER,
f"{secure_filename(project_name)}")
    os.makedirs(extract_dir, exist_ok=True)

    if file_path.endswith(".zip"):
```

```
        shutil.unpack_archive(file_path, extract_dir, "zip")
    elif file_path.endswith(".tar"):
        shutil.unpack_archive(file_path, extract_dir, "tar")
    elif file_path.endswith(".gz"):
        shutil.unpack_archive(file_path, extract_dir, "gztar")
    else:
        raise ValueError("Unsupported archive format")

    return extract_dir


# main page of the website
@views.route('/', methods=['GET', 'POST'])
@login_required
def home():
    return render_template("home.html", user=current_user)


@views.route('/api/project_count')
@login_required
def count_projects():
    count = Project.query.count()
    return jsonify({'count': count})


@views.route('/new_project', methods=['GET', 'POST'])
@login_required
def new_project():
    os.makedirs(UPLOAD_DATASETS_FOLDER, exist_ok=True)

    if request.method == "POST":
        form = request.form
        dataset = None
```

```
        project_name = form.get('project_name')
        classification_classes = get_classification_classes(form.get('classification_classes'))
        classification_method = form.get('classification_method')
        upload_method = form.get('upload_method')
        dataset_link = form.get('dataset_link')
        dataset_file = request.files.get('dataset_archive')

        epochs = 10

        if classification_method == "neural-network":
            epochs = form.get('cnn_epochs')

        error = validate_upload_input(project_name, classification_classes,
                                      classification_method, dataset_file, dataset_link)

        if not error:
            dataset, error = load_dataset_file(upload_method, dataset_file, dataset_link, error,
project_name)

        if not error:
            project = Project(
                project_name=project_name,
                classification_method=classification_method,
                classification_classes=classification_classes,
                dataset=dataset,
                user_id=current_user.id,
                epochs_num=epochs
            )

            db.session.add(project)
            db.session.commit()
```

```
            session['current_project_id'] = project.id

            flash("Project has been successfully created.", category='success')

            return redirect(url_for("views.project_details", project_id=project.id))

    return render_template("home.html", user=current_user)


@views.route('/project/<int:project_id>', methods=['GET', 'POST'])
@login_required
def project_details(project_id):
    project = Project.query.get_or_404(project_id)
    session['current_project_id'] = project_id
    return render_template("project_workspace.html", user=current_user, project=project,
                           class_names=generate_classes_dictionary(project.classification_classes))


@views.route('/create_class_distribution_plot', methods=['POST'])
@login_required
def create_class_distribution_plot():
    return redirect(url_for('views.class_distribution_plot'))


@views.route('/class_distribution_plot', methods=['GET'])
@login_required
def class_distribution_plot():
    return render_plot_page(lambda de, save_path:
de.plot_class_distribution(save_path=save_path),
                            "class_distribution_plot.html", plot_name='class_distribution_plot')


@views.route('/create_sample_spectra_plot', methods=['GET', 'POST'])
```

```
@login_required
def create_sample_spectra_plot():
    if request.method == "POST":
        spectras_to_analyze = request.form.getlist('multiDropdownSpectras')
        session['spectras_to_analyze'] = spectras_to_analyze
    return redirect(url_for('views.sample_spectra_plot'))


@views.route('/sample_spectra_plot', methods=['GET', 'POST'])
@login_required
def sample_spectra_plot():
    selected_classes = session.pop('spectras_to_analyze', None)
    return render_plot_page(lambda de, save_path: de.plot_sample_spectra(save_path=save_path,
                                                                         selected_classes=selected_classes),
                            "sample_spectra_plot.html", plot_name='sample_spectra_plot')


@views.route('/create_comprehensive_review_plot', methods=['POST'])
@login_required
def create_comprehensive_review_plot():
    return redirect(url_for('views.comprehensive_review_plot'))


@views.route('/comprehensive_review_plot', methods=['GET'])
@login_required
def comprehensive_review_plot():
    return render_plot_page(lambda de, save_path:
de.plot_comprehensive_review(save_path=save_path),
                            "comprehensive_review_plot.html",
plot_name="comprehensive_review_plot")


@views.route('/create_classes_prediction_plot', methods=['GET', 'POST'])
@login_required
```

```
def create_classes_prediction_plot():
    # print("create_classes_prediction_plot is called")
    return render_template("loading.html", user=current_user)



@views.route('/classes_prediction_plot', methods=['GET', 'POST'])
@login_required
def classes_prediction_plot():
    # print("classes_prediction_plot is called")
    project, redirect_response = get_current_project_or_redirect()
    if redirect_response:
        return redirect_response

    class_names_dict = generate_classes_dictionary(project.classification_classes)
    gp = None
    method = project.classification_method
    epochs = project.epochs_num

    if request.method == 'POST' and method == "neural-network":
        # print("post")
        try:
            submitted_epochs = int(request.form.get('cnn_epochs'))
            if 1 <= submitted_epochs <= 100:
                epochs = submitted_epochs
                project.epochs_num = epochs
                db.session.commit()
                flash(f"Epochs updated to {epochs}", category="success")
            else:
                flash("Epoch number must be between 1 and 100.", category="error")
        except ValueError:
            flash("Invalid epoch input.", category="error")

    if project.classification_method == "random-forest-classifier":
```

```
    gp = GeneratePredictionRandomForest(class_names=class_names_dict)
    method = "Random Forest Classifier"
elif project.classification_method == "xg-boost":
    gp = GeneratePredictionXGBoost(class_names=class_names_dict)
    method = "XGBoost Classifier"
elif project.classification_method == "k-nearest":
    gp = GeneratePredictionKNearest(class_names=class_names_dict)
    method = "K-Nearest Neighbors"
elif project.classification_method == "logistic-regression":
    gp = GeneratePredictionLogisticRegression(class_names=class_names_dict)
    method = "Logistic Regression"
else:
    print("Number of epochs: ", epochs)
    gp = GeneratePredictionCNN(class_names=class_names_dict, epochs=epochs)
    method = "Convolutional Neural Network"

print(f"loading {method}")

gp.load_data(project.dataset)
image_path, y_pred_img = gp.compute_classification_results()
cm_path = gp.get_confusion_matrix(y_pred_img, method.replace(" ", "_"))

y_test = gp.get_y_test()
pred = gp.get_clf()

present_labels = np.unique(y_test)
target_names = [gp.class_names[k] for k in present_labels]

report_text = classification_report(y_test, pred, target_names=target_names)
report_html = f"<pre>{report_text}</pre>"

print(cm_path)
```

```
    return render_template("class_prediction_plot.html",
                           user=current_user,
                           epochs_num=epochs,
                           project=project,
                           report_html=report_html,
                           classification_method=method,
                           image_path=url_for('static', filename=image_path),
                           cm_path=url_for('static', filename=cm_path))


@views.route('/download', methods=['GET', 'POST'])
def download():
    project, redirect_response = get_current_project_or_redirect()
    if redirect_response:
        return redirect_response

    filename = request.args.get("filename")
    print(filename)

    if not filename:
        flash("Filename not provided", "error")
        return redirect(url_for("views.project_details", project_id=project.id))

    save_path = os.path.join(STATIC_FOLDER, filename.replace("static/", "").replace("\\", "/"))

    if not os.path.exists(save_path):
        flash("File not found", "error")
        return redirect(url_for("views.project_details", project_id=project.id))
    return send_file(save_path, as_attachment=True)


@views.route('/rename-project', methods=['POST'])
def rename_project():
```

```
    data = request.get_json()
    project = json.loads(request.data)
    projectId = project['projectId']
    project = Project.query.get(projectId)
    new_name = data.get('newName')

    if not new_name:
        return jsonify({'success': False, 'message': 'New name is required.'}), 400

    if not project:
        return jsonify({'success': False, 'message': 'Project not found.'}), 404

    if project.user_id != current_user.id:
        return jsonify({'success': False, 'message': 'Unauthorized.'}), 403

    existing_project = Project.query.filter_by(project_name=new_name).first()
    if existing_project:
        return jsonify({'success': False, 'message': 'Project name already exists.'}), 409
    project.project_name = new_name
    db.session.commit()
    return jsonify({'success': True, 'message': 'Project renamed successfully.'})



@views.route('/delete-project', methods=['POST'])
def delete_project():
    project = json.loads(request.data)
    projectId = project['projectId']
    project = Project.query.get(projectId)

    if project:
        if project.user_id == current_user.id:
            db.session.delete(project)
            db.session.commit()
```

```
    return jsonify({})
```

models.py:

```
from flask_login import UserMixin
from sqlalchemy.sql import func
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()

class Project(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    project_name = db.Column(db.String(80), unique=True, nullable=False)
    classification_method = db.Column(db.String(10000), nullable=False)
    classification_classes = db.Column(db.String(10000), nullable=False)
    dataset = db.Column(db.String(255), nullable=False)
    date_created = db.Column(db.DateTime(timezone=True), default=func.now(), nullable=False)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    epochs_num = db.Column(db.Integer, nullable=False)

class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(64), unique=True, nullable=False)
    password = db.Column(db.String(64), nullable=False)
    first_name = db.Column(db.String(64), nullable=False)
    projects = db.relationship('Project', backref='author', lazy=True)
```

auth.py:

```
from flask import Blueprint, render_template, request, flash, redirect, url_for
from .models import User, db
from werkzeug.security import generate_password_hash, check_password_hash
```

```
from flask_login import login_user, login_required, logout_user, current_user

auth = Blueprint('auth', __name__)

@auth.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form.get('email')
        password = request.form.get('password')

        user = User.query.filter_by(email=email).first()

        if user:
            if check_password_hash(user.password, password):
                flash("Logged in successfully!", category='success')
                login_user(user, remember=True)
                return redirect(url_for('views.home'))
            else:
                flash("Wrong password!", category='error')
        else:
            flash("User does not exist!", category='error')
    return render_template("login.html", user=current_user)

@auth.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('auth.login'))


@auth.route('/sign-up', methods=['GET', 'POST'])
def sign_up():
    if request.method == "POST":
```

```
        email = request.form.get('email')
        first_name = request.form.get('firstName')
        password1 = request.form.get('password1')
        password2 = request.form.get('password2')
        user = User.query.filter_by(email=email).first()

        if user:
            flash("User already exists!", category='error')
        elif len(email) < 4:
            flash('Email must be greater than 4 characters.', category='error')
        elif len(first_name) < 2:
            flash('First name must be greater than 2 characters.', category='error')
        elif password1 != password2:
            flash('Passwords do not match.', category='error')
        elif len(password1) < 8:
            flash('Password must be at least 8 characters.', category='error')
        else:
            new_user = User()
            new_user.email = email
            new_user.first_name = first_name
            new_user.password = generate_password_hash(password1)
            db.session.add(new_user)
            db.session.commit()
            login_user(new_user, remember=True)
            flash("Account has been successfully created.", category='success')

            return redirect(url_for('views.home'))
    return render_template("sign_up.html", user=current_user)
```

__init__.py:

```
from Ground_Detection_App.website.views import views
from Ground_Detection_App.website.auth import auth
```

```
from Ground_Detection_App.website.models import User, db
from flask import Flask
from os import path
from flask_login import LoginManager

DB_NAME = "database.db"

# initialize flask
def create_app():
    app = Flask(__name__)
    app.config['SECRET_KEY'] = 'key_halapova'
    app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///' + DB_NAME
    db.init_app(app)

    app.register_blueprint(views, url_prefix='/')
    app.register_blueprint(auth, url_prefix='/')

    with app.app_context():
        db.create_all()

    login_manager = LoginManager()
    login_manager.login_view = "auth.login"
    login_manager.init_app(app)
    @login_manager.user_loader
    def load_user(user_id):
        return User.query.get(int(user_id))

    return app

def create_database(app):
    if not path.exists('instance/' + DB_NAME):
        db.create_all(app=app)
        print('Database created successfully!')
```

```
    else:
        print('Database already exists.')
```

base_classifier.py:

```
import os
import random
import string
import numpy as np
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt
from flask import jsonify
from matplotlib.colors import ListedColormap
from sklearn.decomposition import PCA
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from Ground_Detection_App.website.ml_model.load_data import LoadDataset
matplotlib.use('TkAgg')



class BaseClassifier:

    def __init__(self, class_names):
        self.data = None
        self.labels = None
        self.class_names = class_names
        self.wavelength = None
        self.scaler = None
        self.pca = None
        self.y_test = None
```

```
        self.clf = None
        self.class_colors = sns.color_palette(None, len(class_names))

    def set_y_test(self, y_test):
        self.y_test = y_test

    def get_y_test(self):
        return self.y_test

    def set_clf(self, clf):
        self.clf = clf

    def get_clf(self):
        return self.clf

    def load_data(self, path):
        ld = LoadDataset(self.class_names)
        self.wavelength, self.labels, self.data = ld.load_data(path)

    def preprocess_data(self, pca_components=None, test_size=0.3, random_state=42,
scale=True):
        # min-max normalization [0, 1] assuming a 16-bit range
        data = self.data.astype(np.float32) / 65535.0  # pixel values range from 0 to 65535 in 16-bit
HSI

        #  reshaping the data and reconstructing outputs
        rows, cols, bands = data.shape
        # flatten the data
        X = data.reshape((rows * cols, bands))
        y = self.labels.ravel()
        # Filter out unlabeled pixels. Remove background (unlabeled) pixels
        mask = y > 0
        X = X[mask]
        y = y[mask] - 1
```

```
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size,
random_state=random_state,
                                                            stratify=y)

        if scale:
            self.scaler = StandardScaler()
            X_train = self.scaler.fit_transform(X_train)
            X_test = self.scaler.transform(X_test)

        if pca_components is not None:
            self.pca = PCA(n_components=pca_components)
            X_train = self.pca.fit_transform(X_train)
            X_test = self.pca.transform(X_test)

        return X_train, X_test, y_train, y_test, rows, cols, mask

    @staticmethod
    def save_prediction_to_file(method, metric_type=None):
        current_dir = os.path.dirname(os.path.abspath(__file__))
        static_img_path = os.path.join(current_dir, '..', 'static', 'imgs')
        os.makedirs(static_img_path, exist_ok=True)
        characters = string.ascii_letters + string.digits
        random_string = "".join(random.choice(characters) for _ in range(5))
        if metric_type is not None:
            filename = f'{metric_type}_{method}_{random_string}.png'
        else:
            filename = f'classes_prediction_{method}_{random_string}.png'
        full_img_path = os.path.join(static_img_path, filename)
        plt.savefig(full_img_path)
        plt.close()
        return f'imgs/{filename}'

    def plot_predictions_truth(self, y_pred_img, y_img):
```

```
    plt.figure(figsize=(15, 7))
    cmap = ListedColormap(self.class_colors)
    plt.subplot(1, 2, 1)
    plt.imshow(y_pred_img, cmap=cmap)
    plt.title('Predicted Classes')
    plt.axis('off')
    unique, counts = np.unique(y_pred_img, return_counts=True)
    pixel_counts = dict(zip(unique, counts))
    patches = []
    for i in range(len(self.class_names)):
        class_label = i + 1  # predicted image uses 1-based class indexing
        count = pixel_counts.get(class_label, 0)
        label = f"{i}: {self.class_names[i]}, Pixels: {count}"
        patch = plt.plot([], [], marker="o", ms=10, ls="", mec=None,
                         color=self.class_colors[i], label=label)[0]
        patches.append(patch)
    plt.legend(handles=patches, bbox_to_anchor=(1.05, 1), loc='upper left', fontsize=15)
    plt.subplot(1, 2, 2)
    plt.imshow(y_img, cmap=cmap)
    plt.title('Ground Truths')
    plt.axis('off')
    plt.tight_layout()
    return None

def flattent_data(self, mask, clf, rows, cols, method):
    X_full = self.data.astype(np.float32) / 65535.0
    X_full = X_full.reshape((-1, X_full.shape[-1]))
    X_full_masked = X_full[mask]

    if self.scaler:
        X_full_masked = self.scaler.transform(X_full_masked)

    if self.pca:
```

```
            X_full_masked = self.pca.transform(X_full_masked)

        y_pred_raw = clf.predict(X_full_masked)
        y_pred_full = np.zeros(rows * cols, dtype=np.int32)
        y_pred_full[mask] = y_pred_raw + 1
        y_pred_img = y_pred_full.reshape((rows, cols))
        y_img = self.labels
        self.plot_predictions_truth(y_pred_img, y_img)
        return self.save_prediction_to_file(method), y_pred_img

    def plot_confusion_matrix(self, y_true, y_pred):
        labels = np.unique(y_true)
        cm = confusion_matrix(y_true, y_pred, labels=labels)

        try:
            display_labels = [self.class_names[int(label)] for label in labels]
        except (KeyError, IndexError):
            display_labels = [str(label) for label in labels]

        fig, ax = plt.subplots(figsize=(12, 12))
        disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                                      display_labels=display_labels)
        disp.plot(ax=ax, cmap="Blues", xticks_rotation=45)
        plt.tight_layout()
        return fig

    def get_confusion_matrix(self, y_pred_img, method):
        # flatten both arrays and mask out background if needed
        y_pred_flat = y_pred_img.ravel()
        y_true_flat = self.labels.ravel()
        mask = y_true_flat > 0
        y_pred_flat = y_pred_flat[mask] - 1
        y_true_flat = y_true_flat[mask] - 1
```

```
        self.plot_confusion_matrix(y_true_flat, y_pred_flat)
        return self.save_prediction_to_file(method, metric_type="confusion_matrix")

    def compute_classification_results(self):
        raise NotImplementedError("Must be implemented by subclasses")
```

data_evaluation.py:

```
import os
import random
import string
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
from ..ml_model.load_data import LoadDataset
matplotlib.use('Agg')

class DataEvaluation:

    def __init__(self, class_names):
        self.class_names = class_names
        self.data = None  # hyperspectral image cube
        self.labels = None  # the ground truth classifications
        self.wavelength = None  # the wavelength values for each spectral band

    def load_data(self, path):
        ld = LoadDataset(self.class_names)
        self.wavelength, self.labels, self.data = ld.load_data(path)

    @staticmethod
    def save_plot_to_file(plot_type, save_dir):
```

```
    random_string = "".join(random.choice(string.ascii_letters + string.digits) for _ in range(5))
    full_img_path = os.path.join(save_dir, f'{plot_type}_{random_string}.png')
    plt.savefig(full_img_path)
    plt.close()
    return f'imgs/{plot_type}_{random_string}.png'

def create_false_color(self, bands=None):

    if bands is None:
        bands = [30, 20, 10]
    false_color = np.dstack([self.data[:, :, b] for b in bands])
    return (false_color - false_color.min()) / (false_color.max() - false_color.min())

def plot_class_distribution(self, save_path):

    if self.labels is None:
        raise ValueError("Labels not defined")
    unique, counts = np.unique(self.labels, return_counts=True)
    plt.figure(figsize=(15, 5))
    bars = plt.bar(
        [self.class_names.get(i, f'Unknown material ({i})') for i in unique],
        counts
    )

    plt.xticks(rotation=45, ha='right')
    plt.title('Class Distribution')
    plt.xlabel('Class')
    plt.ylabel('Pixels number')
    for bar in bars:
        plt.text(bar.get_x() + bar.get_width() / 2.,
                 bar.get_height(),
                 f'{int(bar.get_height())}',
                 ha='center', va='bottom'
```

```
            )
        plt.tight_layout()
        static_img_path = os.path.dirname(save_path) if save_path else None
        return self.save_plot_to_file('class_distribution', save_dir=static_img_path)

    # Візуалізація спектральних характеристик
    def plot_sample_spectra(self, save_path, selected_classes=None):

        if not selected_classes or selected_classes is None:
            selected_classes = [i for i in range(len(self.class_names))]

        selected_classes = list(map(int, selected_classes))
        plt.figure(figsize=(20, 10))

        for class_id in selected_classes:
            # маска, тобто обрані елементи для класифікації
            mask = self.labels == class_id
            if np.any(mask):
                # обчислення середнього спектру для всіх пікселів, що задовольняють умови, визначені маскою
                mean_spectrum = np.mean(self.data[mask], axis=0)
                plt.plot(self.wavelength, mean_spectrum, label=self.class_names[class_id])

        # Середні спектри обраних класів
        plt.title('Selected Classes Average Spectrum')
        plt.xlabel('Spectra Wavelength (nm)')
        # Відбивна здатність матеріалу
        plt.ylabel('Object Reflectivity')
        plt.legend()
        plt.grid(True)

        static_img_path = os.path.dirname(save_path) if save_path else None
```

```
        return self.save_plot_to_file('sample_spectra', save_dir=static_img_path)

    def plot_comprehensive_review(self, save_path):
        fig, axes = plt.subplots(2, 2, figsize=(15, 15))

        false_color = self.create_false_color()
        axes[0, 0].imshow(false_color)
        axes[0, 0].set_title('Colored picture')

        # Ground truth
        im = axes[0, 1].imshow(self.labels)
        axes[0, 1].set_title('Ground Truth')
        plt.colorbar(im, ax=axes[0, 1])

        # Single band visualization
        mid_band = self.data.shape[2] // 2
        im = axes[1, 0].imshow(self.data[:, :, mid_band])
        axes[1, 0].set_title(f'Separate spectral channel ({int(self.wavelength[mid_band])}nm)')
        plt.colorbar(im, ax=axes[1, 0])

        spectral_variance = np.std(self.data, axis=2)
        im = axes[1, 1].imshow(spectral_variance)
        axes[1, 1].set_title('Spectral dispersion')
        plt.colorbar(im, ax=axes[1, 1])
        plt.subplots_adjust(wspace=0.4, hspace=0.4)
        static_img_path = os.path.dirname(save_path) if save_path else None
        return self.save_plot_to_file('comprehensive_review', save_dir=static_img_path)
```

k_nearest_classes_prediction.py:

```
import numpy as np
from sklearn.metrics import classification_report
from sklearn.neighbors import KNeighborsClassifier
```

```
from Ground_Detection_App.website.ml_model.base_classifier import BaseClassifier

class GeneratePredictionKNearest(BaseClassifier):

    def __init__(self, class_names):
        super().__init__(class_names)

    def compute_classification_results(self):
        X_train, X_test, y_train, y_test, rows, cols, mask = self.preprocess_data(
            pca_components=None,
            scale=True,
            test_size=0.3,
            random_state=42
        )
        knn = KNeighborsClassifier(n_neighbors=len(self.class_names))

        knn.fit(X_train, y_train)
        self.set_y_test(y_test)
        self.set_clf(knn.predict(X_test))
        return self.flattent_data(mask, knn, rows, cols, 'knn')
```

load_data.py:

```
import glob
import os
import shutil
import matplotlib
import numpy as np
import scipy.io as sio
import kagglehub
matplotlib.use('Agg')

class LoadDataset:
```

```
    def __init__(self, class_names):
        self.class_names = class_names
        self.data = None
        self.labels = None
        self.wavelength = None

    def load_mat_data(self, path):
        mat_files = glob.glob(os.path.join(path, "*.mat"))

        self.data = None
        self.labels = None

        for mat_file in mat_files:
            mat = sio.loadmat(mat_file)
            for key, value in mat.items():
                if isinstance(value, np.ndarray):
                    if value.ndim == 3 and self.data is None:
                        self.data = value
                    elif value.ndim == 2 and np.issubdtype(value.dtype, np.integer) and self.labels is
None:
                        self.labels = value

        if self.data is None or self.labels is None:
            raise ValueError("Could not automatically find data and label arrays.")

    def load_numpy_data(self, path):
        npy_data = glob.glob(os.path.join(path, '*.npy'))

        self.data = None
        self.labels = None

        for file in npy_data:
```

```
            arr = np.load(file)
            if arr.ndim == 3 and self.data is None:
                self.data = arr
            elif arr.ndim == 2 and np.issubdtype(arr.dtype, np.integer) and self.labels is None:
                self.labels = arr

        if self.data is None or self.labels is None:
            raise ValueError('No data loaded')

    def load_npz_data(self, path):
        npz_data = np.load(path)
        self.data = None
        self.labels = None
        for key in npz_data:
            arr = npz_data[key]
            if arr.ndim == 3 and self.data is None:
                self.data = arr
            elif arr.ndim == 2 and np.issubdtype(arr.dtype, np.integer) and self.labels is None:
                self.labels = arr

        if self.data is None or self.labels is None:
            raise ValueError('No data loaded')

    @staticmethod
    def unarchive_data(archive_format, path):
        filename = os.path.join('Ground_Detection_App', 'website', 'static', 'uploads', path)
        extract_dir = os.path.join('Ground_Detection_App', 'website', 'static', 'uploads')

        shutil.unpack_archive(filename, extract_dir, archive_format)

    def load_data(self, path):

        if "static\\uploads\\" not in path:
```

```
            path = eval(path)

        if os.path.isdir(path):
            if any(f.endswith('.mat') for f in os.listdir(path)):
                self.load_mat_data(path)
            elif any(f.endswith('.npy') for f in os.listdir(path)):
                self.load_numpy_data(path)
            else:
                raise ValueError("Unsupported files in folder.")
        elif path.endswith('.npz'):
            self.load_npz_data(path)
        else:
            raise ValueError("Unsupported file format.")
        self.wavelength = np.linspace(430, 860, self.data.shape[2])
        return self.wavelength, self.labels, self.data
```

logistic_regression_classes_prediction.py:

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from Ground_Detection_App.website.ml_model.base_classifier import BaseClassifier

class GeneratePredictionLogisticRegression(BaseClassifier):

    def __init__(self, class_names):
        super().__init__(class_names)

    def compute_classification_results(self):
        X_train, X_test, y_train, y_test, rows, cols, mask = self.preprocess_data(
            pca_components=None,
            scale=True,
            test_size=0.3,
```

```
            random_state=42
        )
        clf = LogisticRegression(max_iter=1000, random_state=42)
        clf.fit(X_train, y_train)

        self.set_y_test(y_test)
        self.set_clf(clf.predict(X_test))
        return self.flattent_data(mask, clf, rows, cols, 'logistic-regression')
```

neural_network_classes_prediction.py:

```
import random
import numpy as np
from keras import Sequential
from keras.src.layers import Dense, Flatten, Conv2D, MaxPooling2D
from keras.src.optimizers import Adam
from keras.src.utils import to_categorical
from scipy.ndimage import interpolation
from sklearn.decomposition import PCA
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from Ground_Detection_App.website.ml_model.base_classifier import BaseClassifier

class GeneratePredictionCNN(BaseClassifier):

    def __init__(self, class_names, epochs):
        super().__init__(class_names)
        self.numComponents = 50
        self.patch_size = 7
        self.test_size = 0.3
        self.epochs = epochs
```

```
    @staticmethod
    def apply_pca(X, numComponents=75):
        # reducing dimensionality
        newX = X.reshape((-1, X.shape[-1]))
        pca = PCA(n_components=numComponents, whiten=True)
        newX = pca.fit_transform(newX)

        # reshaping back to 3D array
        newX = np.reshape(newX, (X.shape[0], X.shape[1], numComponents))

        return newX, pca

    @staticmethod
    def pad_with_zeros(X, margin=2):
        newX = np.zeros((X.shape[0] + 2 * margin, X.shape[1] + 2 * margin, X.shape[2]))
        x_offset = margin
        y_offset = margin
        newX[x_offset:X.shape[0] + x_offset, y_offset:X.shape[1] + y_offset, :] = X
        return newX

    def create_patches(self, X, removeZeroLabels=True):
        y = self.labels
        margin = int((self.patch_size - 1) / 2)
        zeroPaddedX = self.pad_with_zeros(X, margin=margin)
        patchesX = np.zeros((X.shape[0] * X.shape[1], self.patch_size, self.patch_size,
X.shape[2]))
        patchesY = np.zeros((X.shape[0] * X.shape[1]))

        patchIndex = 0

        for row in range(margin, zeroPaddedX.shape[0] - margin):
            for col in range(margin, zeroPaddedX.shape[1] - margin):
                patch = zeroPaddedX[row - margin:row + margin + 1, col - margin:col + margin + 1]
```

```
                patchesX[patchIndex, :, :, :] = patch
                patchesY[patchIndex] = y[row - margin, col - margin]
                patchIndex = patchIndex + 1
        if removeZeroLabels:
            patchesX = patchesX[patchesY > 0, :, :, :]
            patchesY = patchesY[patchesY > 0]
            patchesY -= 1

        return patchesX, patchesY

    @staticmethod
    def augment_data(X_train):
        flipped_patch = None
        for i in range(int(X_train.shape[0] / 2)):
            patch = X_train[i, :, :, :]
            num = random.randint(0, 2)
            if num == 0:
                flipped_patch = np.flipud(patch)
            if num == 1:
                flipped_patch = np.fliplr(patch)
            if num == 2:
                no = random.randrange(-180, 180, 30)
                flipped_patch = interpolation.rotate(patch, no, axes=(1, 0), reshape=False,
output=None,
                                                     order=3, mode='constant', cval=0.0, prefilter=False)
            patch2 = flipped_patch
            X_train[i, :, :, :] = patch2
        return X_train

    def patch(self, data, height_index, width_index):
        height_slice = slice(height_index, height_index + self.patch_size)
        width_slice = slice(width_index, width_index + self.patch_size)
        patch = data[height_slice, width_slice, :]
```

```
        return patch


    def train_model(self, X_train, y_train):


        model = Sequential()
        model.add(
            Conv2D(32, (3, 3), activation='relu', input_shape=(self.patch_size, self.patch_size,
self.numComponents)))
        model.add(MaxPooling2D((2, 2)))
        model.add(Flatten())
        model.add(Dense(128, activation='relu'))
        model.add(Dense(y_train.shape[1], activation='softmax'))
        model.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=1e-4),
metrics=['accuracy'])
        model.summary()
        model.fit(X_train, y_train, batch_size=32, epochs=self.epochs, verbose=1)
        return model


    def generate_classified_image(self, X, model):


        height = self.labels.shape[0]
        width = self.labels.shape[1]
        patches = []
        positions = []
        outputs = np.zeros((height, width))


        margin = self.patch_size // 2
        for i in range(margin, height - margin):
            for j in range(margin, width - margin):
                target = int(self.labels[i, j])
                if target == 0:
                    continue
                image_patch = self.patch(X, i - margin, j - margin)
                patches.append(
```

```
                image_patch.reshape(1, self.patch_size, self.patch_size,
self.numComponents).astype('float32'))
            positions.append((i, j))

    patches = np.concatenate(patches, axis=0)
    predictions = model.predict(patches)
    for prediction, position in zip(predictions, positions):
        outputs[position[0]][position[1]] = np.argmax(prediction) + 1

    return np.array(outputs.astype(int))

  def compute_classification_results(self):

    X, pca = self.apply_pca(self.data, numComponents=self.numComponents)
    XPatches, yPatches = self.create_patches(X)
    X_train, X_test, y_train, y_test = train_test_split(XPatches, yPatches,
test_size=self.test_size,
                                                        random_state=345, stratify=yPatches)
    X_train = self.augment_data(X_train)
    X_train = X_train.reshape(X_train.shape[0], -1)
    y_train = to_categorical(y_train)

    X_train = X_train.reshape(-1, self.patch_size, self.patch_size, self.numComponents)
    model = self.train_model(X_train, y_train)
    y_pred_img = self.generate_classified_image(X, model)
    self.plot_predictions_truth(y_pred_img, self.labels)
    present_labels = np.unique(y_test)
    target_names = [self.class_names[k] for k in present_labels]
    y_pred_proba = model.predict(X_test)  # ймовірності
    y_pred = np.argmax(y_pred_proba, axis=1)  # індекси найбільших значень → класи

    self.set_y_test(y_test)
    self.set_clf(y_pred)
    return self.save_prediction_to_file("cnn"), y_pred_img
```

random_forest_classes_prediction.py:

```
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from Ground_Detection_App.website.ml_model.base_classifier import BaseClassifier

class GeneratePredictionRandomForest(BaseClassifier):

    def __init__(self, class_names):
        super().__init__(class_names)

    def compute_classification_results(self):
        X_train, X_test, y_train, y_test, rows, cols, mask = self.preprocess_data(
            pca_components=None,
            scale=True,
            test_size=0.3,
            random_state=42
        )
        clf = RandomForestClassifier(n_estimators=100, random_state=42)
        clf.fit(X_train, y_train)
        self.set_y_test(y_test)
        self.set_clf(clf.predict(X_test))
        return self.flattent_data(mask, clf, rows, cols, 'random_forest')
```

xg_boost_classes_prediction.py:

```
import numpy as np
from sklearn.metrics import classification_report
from xgboost import XGBClassifier

from Ground_Detection_App.website.ml_model.base_classifier import BaseClassifier
```

```
class GeneratePredictionXGBoost(BaseClassifier):

    def __init__(self, class_names):
        super().__init__(class_names)

    def compute_classification_results(self, n_iterations=5):
        X_train, X_test, y_train, y_test, rows, cols, mask = self.preprocess_data(
            pca_components=100,
            test_size=0.3,
            random_state=42,
            scale=False
        )
        clf = XGBClassifier(n_estimators=100, max_depth=5)
        clf.fit(X_train, y_train)
        self.set_y_test(y_test)
        self.set_clf(clf.predict(X_test))
        return self.flattent_data(mask, clf, rows, cols, "xg-boost")
```

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Кваліфікаційна робота на тему: "Веб-ресурс для аналізу та класифікації земних матеріалів на основі гіперспектрального аналізу"

ДУІКТ

Виконала: здобувачка вищої освіти гр. ШІД-41
Софія ХАЛАПОВА
Керівник: доктор технічних наук, професор
Андрій БОНДАРЧУК

Мета роботи: розробка інтерактивного веб-застосунку для аналізу гіперспектральних зображень

Об'єкт дослідження: процес класифікації матеріалів зондування землі на основі гіперспектрального аналізу.

Предмет дослідження: сукупність технологій створення веб-ресурсу та методи машинного навчання для аналізу та класифікації об'єктів на гіперспектральних зображеннях.

Основні завдання кваліфікаційної роботи:

1. Аналіз предметної області,
2. Аналіз засобів реалізації веб-застосунку,
3. Проєктування та реалізація веб-застосунку з інтегруванням методів машинного навчання.

Актуальність теми

Аналіз Гіперспектральних зображень застосовують у сільському господарстві, екології, військовій сфері та геології. Алгоритми ШІ здатні здатні автоматизувати обробку великих масивів HSI даних та формувати інтерпретовані висновки з них.

Рис. 1.1 - IBM Watson Studio

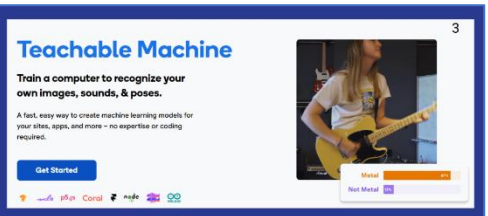


Рис. 1.2 - Teachable Machine

Існують платформи для класифікації RGB зображень, такі як Teachable Machine від Google, IBM Watson Studio чи Microsoft Custom Vision, однак галузь обробки HSI зображень не заповнена, що підкреслює важливість впровадження даного проєкту.

Аналіз наявних рішень

Використані технології:

- Flask;
- SQLite;
- XGBoost;
- Random Forest Classifier;
- K-nearest;
- Логістична регресія;
- Згорткова нейронна мережа.

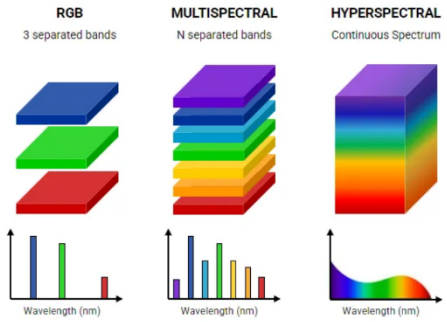


Рис. 1.3 - розподіл спектрів у зображеннях

Гіперспектральна (HSI) зйомка знаходиться у діапазоні від ультрафіолету до ближнього інфрачервоного спектру (300–2500 нм). Гіперспектральні датчики забезпечують високу спектральну роздільну здатність, що дає змогу ідентифікувати матеріали, визначати їхній хімічний склад, текстуру та інші фізичні властивості.

XGBoost, як метод градієнтного бустингу, демонструє гнучкість і потужність у навчанні послідовності слабких моделей. Його ефективність полягає у здатності мінімізувати помилки попередніх прогнозів, створюючи точніші дерева рішень.

$$SS = \frac{(\sum residuals)^2}{\sum P_{i-1} \cdot (1 - P_{i-1}) + \lambda}$$

Формула 1.1 - Similarity Score

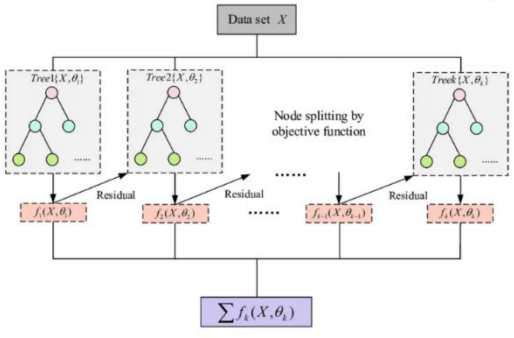


Рис. 1.4 - Схематичне зображення роботи XGBoost

Передбачення XGBoost

Рис. 1.5 - Класифікація XGBoost використовуючи датасет “Pavia University HSI”

RandomForestClassifier навчається із застосуванням балансування класів, що забезпечує рівномірний розподіл даних між класами. Алгоритм Random Forest базується на ансамблевому методі навчання, який використовує множину дерев рішень (Decision Trees) для створення остаточного прогнозу.

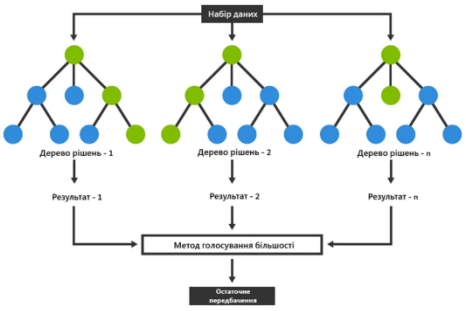


Рис. 1.8 - Схематичне зображення роботи Random Forest

$$Y_{final} = \arg\max_{k} \sum_{i=1}^{N} I(y_i = k)$$

Формула 1.2 - Majority Voting

Передбачення Random Forest

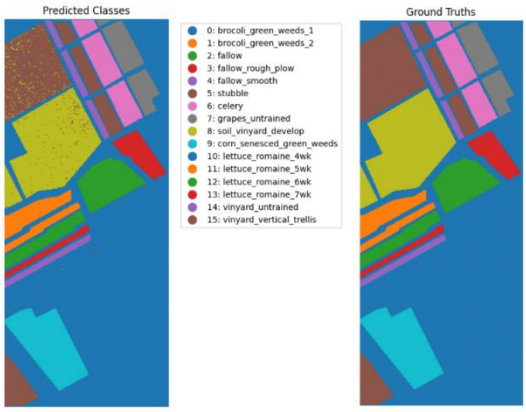


Рис. 1.10 - Класифікація Random Forest використовуючи датасет “Salinas”

Метод K-найближчих сусідів це алгоритм машинного навчання з учителем для задач класифікації, а також регресії. Суть даного алгоритму полягає у розпізнаванні об'єктів, які знаходяться поруч одне з одним, тобто знаходження «сусідів». Метод передбачає, що предмети, які лежать поруч ймовірно мають схожі значення, ознаки, характеристики, параметри, тобто відносяться до одного класу.

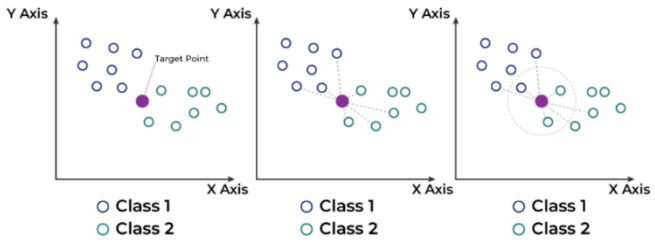


Рис. 1.12 - Схематичне зображення роботи K-nearest

Передбачення K-nearest

Рис. 1.15 - Класифікація K-nearest використовуючи датасет "Indian Pines"

Біноміальна LogReg - бінарна класифікація. LogReg використовує сигмоїдну ф-цію. Ф-ція визначає ймовірності, які лежать у діапазоні від 0 до 1.

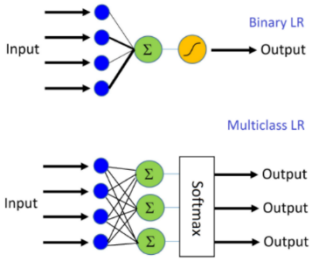


Рис. 1.16- Схематичне зображення роботи LogReg

$$f(x) = \frac{1}{1+e^{-x}}$$

Формула 1.3 - Sigmoid Function

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Формула 1.4 - Softmax

Мультиноміальна LogReg - класифікація багатьох класів. Замість сигмоїди - ф-ція Softmax, що приймає вектор значень та перетворює його на вектор ймовірностей.

Передбачення Logistic Regression

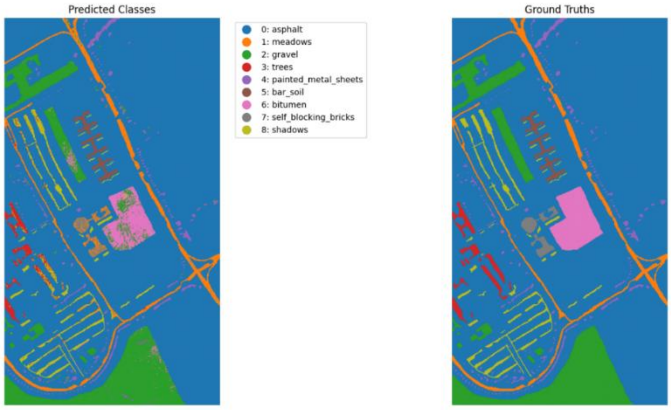


Рис. 1.17 - Класифікація Log-Reg використовуючи датасет "Pavia University HSI"

CNN ефективні для гіперспектральних даних, оскільки вони аналізують зображення не лише як візуальні форми, а й як багатошарові спектральні дані.

Шари CNN:

1. *Conv2D* (32 фільтри, ядро 3×3).
2. *Функція активації:* ReLU.
3. *MaxPooling2D* (2×2).
4. *Flatten.*
5. *Dense* (128 нейронів).
6. *Dense* (N класів).
7. *Активація softmax*

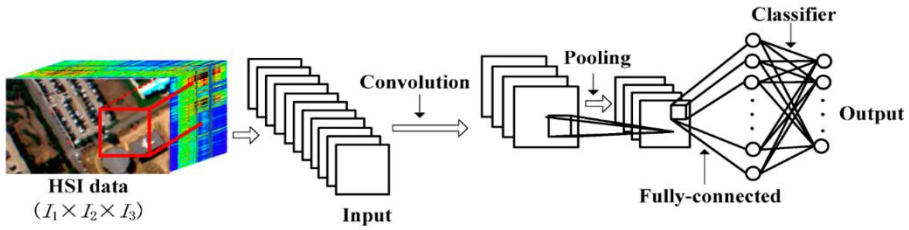


Рис. 1.20 - Схематичне зображення роботи Згорткової Нейронної Мережі

Передбачення Convolutional neural network

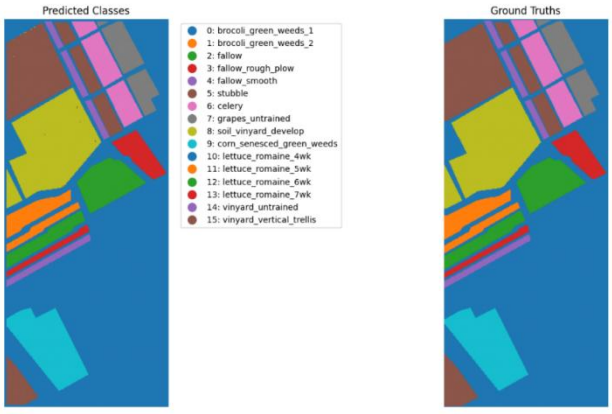


Рис. 1.22 - Класифікація CNN використовуючи датасет "Salinas"

Метод	Accuracy	Precision (Macro)	Recall (Macro)	F1-score (Macro)	Precision (Weighted)	Recall (Weighted)	F1-score (Weighted)
XGBoost	0.96	0.98	0.98	0.98	0.96	0.96	0.96
Log Reg	0.93	0.97	0.96	0.97	0.93	0.93	0.93
Random Forest	0.95	0.98	0.97	0.97	0.95	0.95	0.95
KNN	0.87	0.92	0.92	0.92	0.87	0.87	0.86
CNN	0.99	0.99	0.99	0.99	0.99	0.99	0.99

Де,

- ***Macro avg*** - середнє значення по класах
- ***Weighted avg*** - зважене середнє значення згідно ваг
- ***Accuracy*** - загальна точність моделі
- ***Precision*** - точність передбачень
- ***Recall*** - повнота виявлення класів
- ***F1-score*** - баланс точності й повноти класифікації

Порівняльна характеристика ML методів для класифікації HSI

Принцип роботи програми

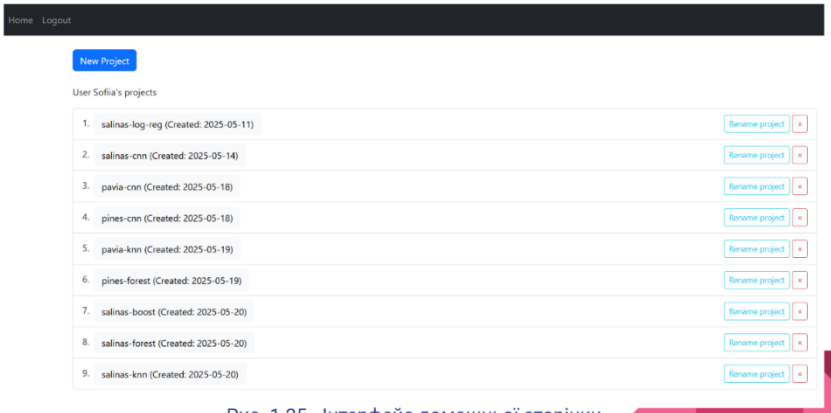


Рис. 1.25 - Інтерфейс домашньої сторінки користувача

Принцип роботи програми

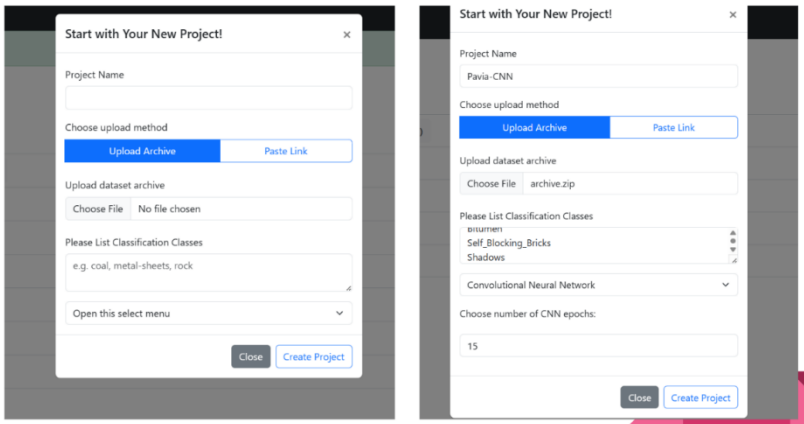
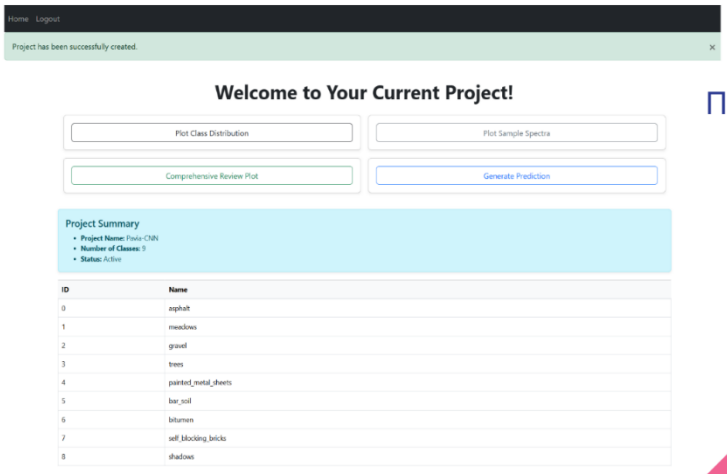
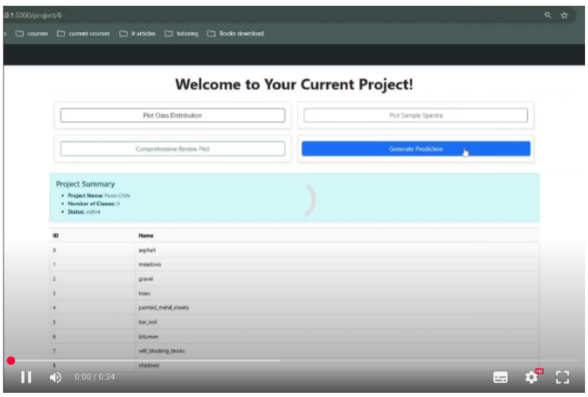


Рис. 1.26 - Інтерфейс створення ML проєкту

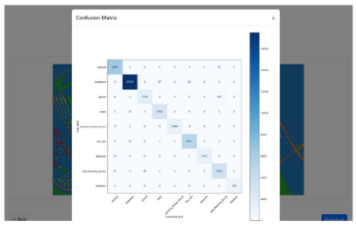


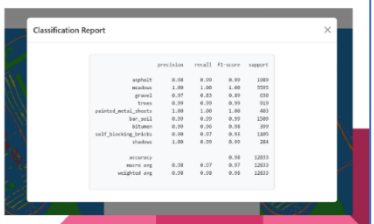
Принцип роботи програми

Рис. 1.27 - Домашня сторінка проєкту



Відео. 1.1- Принцип роботи класифікації





Висновки:

Отже, у цьому проєкті було розроблено веб-застосунок для класифікації матеріалів на гіперспектральних зображеннях із використанням сучасних ML алгоритмів. Розробка стала прикладом успішного поєднання гіперспектрального аналізу та машинного навчання у зручному для користувача форматі, що має перспективи для подальшого розвитку та впровадження.

Дякую за увагу!