Міністерство освіти і науки України
Національний технічний університет «Дніпровська політехніка»

Інститут електроенергетики
(навчально-науковий інститут)
Факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня магістр
(бакалавра, магістра)

Здобувача вищої освіти Носуля Олега Павловича
(ПІБ)

академічної групи 126м-23-1
(шифр)

спеціальності 126 «Інформаційні системи та технології»
(код і назва спеціальності)

спеціалізації за освітньо-професійною (освітньо-науковою) програмою
(за наявності)
«Інформаційні системи та технології»
(офіційна назва)

на тему Інформаційна технологія розпізнавання та класифікації рослин на основі аналізу зображення
(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	Булана Т.М.			

Рецензент	Клеменко А.В.			

Нормоконтролер	Коротенко Г.М.			

Дніпро
2024

ЗАТВЕРДЖЕНО:
завідувач кафедри
інформаційних технологій та комп'ютерної інженерії
(повна назва)

_____ __В.В. Гнатушенко____
(підпис) (ініціали та прізвище)

«_____» _____20___року

ЗАВДАННЯ

на кваліфікаційну роботу
ступеня_____магістр_____
(бакалавра, магістра)

здобувача вищої освіти_____Носуля О.П.____**академічної групи**___126м-23-1____
(прізвище та ініціали) (шифр)

спеціальності_____126 «Інформаційні системи та технології»_____

спеціалізації за освітньою-професійною програмою «Інформаційні системи та технології»
(за наявності)

на тему__ Інформаційна технологія розпізнавання та класифікації рослин на основі аналізу зображення

затверджену наказом ректора НТУ «Дніпровська політехніка» від 17.10.2024 № 1388-С

Розділ	Зміст	Термін виконання
Розділ 1	Аналіз існуючих рішень	20.09.2024-01.12.2024
Розділ 2	Використані математичні моделі	30.09.2024-15.10.2024
Розділ 3	Опис програмного забезпечення та технологій	15.10.2024-05.11.2024
Розділ 4	Апробація та тестування	05.11.2024-10.11.2024

Завдання видано _____ _____Булана Т.М._____
(підпис керівника) (ініціали та прізвище)

Дата видачі 30.08.2024 р.

Дата подання до екзаменаційної комісії_____

Прийнято до виконання _____ ___Носуль О.П____
(підпис здобувача вищої освіти) (ініціали та прізвище)

РЕФЕРАТ

Пояснювальна записка: 88 стор., 27 рис., 4 додатки, 66 джерел.

Об'єкт дослідження: методи використання нейронних мереж для аналізу зображень у сільському господарстві.

Предмет дослідження: автоматизована система діагностики стану рослин на основі нейронних мереж.

Мета магістерської роботи: розробка інформаційної системи для автоматизованого аналізу зображень рослин, яке дозволяє виявляти захворювання.

У вступі описано актуальність теми дослідження, виконано постановку задачі, а також наведено огляд сучасних підходів до автоматизації агропромислових процесів.

Перший розділ присвячено аналізу існуючих рішень у сфері автоматизованого моніторингу стану рослин. Розглянуто природу даних і типи зображень (RGB, монохромні, інфрачервоні), їх застосування для діагностики, а також популярні джерела отримання зображень: камери, супутники, дрони. Проведено огляд методів класифікації, детекції та сегментації зображень, таких як CNN, YOLO та Mask R-CNN.

Другий розділ детально описує методи побудови наборів даних. Розглянуто процес підготовки зображень для аналізу, їх структурування, анотацію за допомогою LabelImg, а також особливості організації даних для навчання нейронних мереж. Проаналізовано формати анотацій (YOLO, Pascal VOC) і вибір архітектури YOLOv8 для задачі.

Третій розділ описує математичні моделі та архітектури нейронних мереж, що використовуються для аналізу зображень рослин. Висвітлено особливості структури даних, ієрархію пікселів і об’єктів, а також деталі роботи сучасних моделей, таких як U-Net, ResNet, Faster R-CNN і Vision Transformer. Порівняно ефективність різних підходів для класифікації, детекції та сегментації.

Четвертий розділ присвячено реалізації інформаційної системи та аналізу отриманих результатів. Описано процес навчання нейронної мережі, тестування

та оцінки її ефективності за ключовими метриками (IoU, Precision, Recall). Розглянуто можливості практичного застосування системи в агросекторі.

Практична цінність роботи полягає у створенні інформаційної системи, що дозволяє автоматизувати процес діагностики захворювань рослин, знижуючи витрати на моніторинг та догляд, а також підвищуючи врожайність. Розроблене рішення може бути застосоване як у промислових агрокомплексах, так і в домашньому садівництві.

НЕЙРОННІ МЕРЕЖІ, YOLOV8, АНАЛІЗ ЗОБРАЖЕНЬ, ДІАГНОСТИКА РОСЛИН, СІЛЬСЬКЕ ГОСПОДАРСТВО, АВТОМАТИЗАЦІЯ.

ABSTRACT

Explanatory note: 88 pages, 27 figures, 4 appendices, 66 sources.

Object of research: Methods of using neural networks for image analysis in agriculture.

Subject of research: Automated plant condition diagnostics system based on neural networks.

The purpose of the master`s work: Development of software for automated plant image analysis, enabling the detection of diseases and providing treatment recommendations.

The introduction describes the relevance of the research topic, defines the task, and provides an overview of modern approaches to automating agribusiness processes.

The first section is dedicated to analyzing existing solutions in the field of automated monitoring of plant conditions. It explores the nature of data and types of images (RGB, monochrome, infrared), their applications for diagnostics, and popular image acquisition sources: cameras, satellites, drones. It provides an overview of image classification, detection, and segmentation methods, such as CNN, YOLO, and Mask R-CNN.

The second section details methods for creating datasets. It discusses the process of preparing images for analysis, structuring them, annotating them using LabelImg, and organizing data for training neural networks. Annotation formats (YOLO, Pascal VOC) and the choice of the YOLOv8 architecture for the task are analyzed.

The third chapter describes the mathematical models and neural network architectures used for plant image analysis. It highlights the characteristics of data structure, the hierarchy of pixels and objects, and the operational details of modern models such as U-Net, ResNet, Faster R-CNN, and Vision Transformer. The chapter compares the efficiency of different approaches for classification, detection, and segmentation.

The fourth section focuses on software implementation and analysis of the obtained results. It describes the process of training the neural network, testing, and evaluating its efficiency based on key metrics (IoU, Precision, Recall). It also discusses

the practical application of the system in the agricultural sector.

The practical value of the work is the creation of a tool that automates the plant disease diagnostics process, reducing costs for monitoring and care, as well as increasing crop yield. The developed solution can be applied in both industrial agro-complexes and home gardening.

NEURAL NETWORKS, YOLOV8, IMAGE ANALYSIS, PLANT DIAGNOSTICS, AGRICULTURE, AUTOMATION.

ЗМІСТ

Перелік умовних позначень

CNN – Convolutional Neural Network – згорткова нейронна мережа

RNN – Recurrent Neural Network – рекурентна нейронна мережа

YOLO – You Only Look Once – алгоритм детекції об’єктів

YOLOv8 – остання версія алгоритму YOLO

ViT – Vision Transformer – трансформер для обробки зображень

SSD – Single Shot MultiBox Detector – алгоритм одноетапної детекції об'єктів

FPN – Feature Pyramid Network – мережа пірамідальних ознак

PANet – Path Aggregation Network – мережа агрегації шляхів

GPU – Graphics Processing Unit – графічний процесор

CPU – Central Processing Unit – центральний процесор

IoU – Intersection over Union – міра перекриття між об'єктами

SGD – Stochastic Gradient Descent – стохастичний градієнтний спуск

API – Application Programming Interface – інтерфейс прикладного програмування

XML – Extensible Markup Language – розширювана мова розмітки

Pascal VOC – стандарт формату для анотацій об'єктів на зображеннях

PyTorch – фреймворк для машинного навчання

Ultralytics – бібліотека для роботи з YOLO

LabelImg – інструмент для ручної розмітки зображень

NMS – Non-Maximum Suppression – метод пригнічення непотрібних передбачень

SE – Squeeze-and-Excitation – метод активації важливих ознак

ВСТУП

Сільське господарство завжди буде однією з ключових сфер життя людства, забезпечуючи потребу в продуктах харчування та сировині. З плином часу ця галузь зазнала значних змін, спрямованих на підвищення ефективності, скорочення втрат і оптимізацію ресурсів. Сьогодні, у час стрімкого розвитку інформаційних технологій, зокрема штучного інтелекту, з'являються нові можливості для інтеграції передових технологій у аграрний сектор.

Однією з важливих проблем сучасного сільського господарства є своєчасне виявлення та лікування хвороб рослин. Традиційні методи моніторингу здоров'я рослин часто потребують значних людських ресурсів, часу та спеціалізованих знань. Використання методів машинного навчання, таких як нейронні мережі, відкриває нові перспективи для автоматизації цих процесів. Нейронні мережі здатні аналізувати зображення рослин, визначати їх стан і локалізувати проблемні ділянки з високою точністю.

Метою даної роботи є розробка інформаційної системи для автоматизованого аналізу зображень рослин із застосуванням нейронних мереж. Такий підхід дозволить аграріям і дослідникам своєчасно виявляти хвороби, знижувати витрати на моніторинг та покращувати якість і кількість урожаю. Використання інструментів комп’ютерного зору, таких як YOLO, Faster R-CNN, U-Net і Vision Transformer, забезпечить високу точність і швидкість обробки даних, що є критично важливим для практичного застосування.

Пропоноване рішення може знайти застосування в різних умовах - від теплиць і невеликих фермерських господарств до великих агропромислових комплексів.

Для досягнення мети у даній роботі необхідно вирішити такі завдання:

1) дослідити сучасні методи та підходи до аналізу зображень рослин за допомогою нейронних мереж;
2) розробити структуру та підготувати набір даних для навчання моделей;
3) створити інформаційну систему, яка реалізує алгоритми аналізу зображень;

4) провести тестування та оцінити ефективність запропонованої системи на практичних прикладах.

Інтеграція інноваційних технологій у сільське господарство відкриває нові можливості для підвищення продуктивності, стійкості та конкурентоспроможності аграрного сектору. Робота спрямована на розвиток ефективних рішень, які сприятимуть оптимізації виробничих процесів і забезпечать стабільний розвиток галузі в умовах глобальних викликів.

РОЗДІЛ 1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1 Об'єкт дослідження та предмета область

1.1.1 Природа даних та типи зображень

Об’єктом дослідження цієї роботи є методи та технології використання нейронних мереж з ціллю автоматизованого аналізу зображень рослин з метою їх діагностики. Що включає у себе вивчення можливості застосування комп’ютерного зору для виявлення захворювань рослин на основі різноманітних типів зображень, зокрема RGB (червоний-зелений-синій), монохромних (чорно-білі), або інфрачервоних зображень, що отримуватимуться через різноманітні джерела.

Вибір типу зображень маж дуже важливе значення для ефективності діагностики стану рослин. В основному для автоматизованого аналізу використовуються наступні типи зображень:

1) RGB зображення. Це стандартний формат зображень що містить 3 кольорових каналів (червоний-зелений-синій), що у комбінації дають повну картину кольорів. RGB є найбільш поширеним типом у сфері комп’ютерного зору для виявлення хвороб рослин оскільки більшість захворювань рослин впливають на колір рослин, що дозволяє легко ідентифікувати проблеми через зміну кольору листя, стеблів, тощо. Ці зображення можуть мати велику розподільну знатність, що дозволяє детально аналізувати кожен піксель враховуючи дрібні зміни в текстурі та кольорі.
2) Монохромні зображення. У цьому випадку використовується лише один канал для представлення. Потрібно це у ситуаціях коли кольорові зміни не є ключовими і може бути корисним у випадку якщо важливіше виділяти контури, або структур об’єкту. Цей тип часто використовується з ціллю сегментації.
3) Інфрачервоні зображення (ІЧ). Використовуються для виявлення температурних змін на поверхні рослин. Може бути особливо корисним

для моніторингу стану рослин у нічний час або в умовах низької освітленості так як ІЧ спектр не залежить від видимого спектру світла. Дозволяє виявити відмінності рослин оскільки хвороби або стресові стани можуть призводити до змін температури поверхні рослин. [1, 2, 3, 4, 5, 6]

1.1.2 Джерела отримання даних

Зображення для аналізу можуть бути отримані через різноманітні джерела, кожне з яких має власні переваги та недоліки в залежності від конкретних умов. [7, 8, 9, 10, 11, 12, 30, 31]

Розглянемо три найпопулярніші джерела для отримання даних:

1) Камери. Найбільш поширене джерело зображень для аналізу. Використовуються як стаціонарні камери так і мобільні пристрої. Дозволяють отримувати високоякісні зображення в різних умовах. Вони також можуть бути оснащені спеціалізованими лінзами для зйомки у різних спектрах, включаючи ІЧ, що дозволяє точно визначити зміни в стані рослин.
2) Супутники. Супутники дозволяють моніторити великі площі, такі як сільськогосподарські угіддя. Вони надають зображення у різних спектрах, включаючи видимий, ІЧ, та навіть УФ. Завдяки супутникам можна здійснювати моніторинг на глобальному рівні, виявляючи проблеми, що виникають на великих площах. Це особливо важливо для аграріїв та дослідників.
3) Дрони оснащені камерами, або сенсорами для високоякісних зйомок зображень з висоти, що дозволяє отримати дані з обмежених територій або з важкодоступних ділянок. Використання дронів для моніторингу стану рослин так сільськогосподарських культур дозволяє отримувати зображення високої розподільної здатності для подальшого виявлення пошкоджень або хвороб рослин.

1.1.3 Методи розпізнавання

Методи для розпізнавання, що базуються на алгоритмах машинного навчання, зокрема нейронних мережах, дозволяють автоматизовано визначати стан рослин за допомогою аналізу отриманих зображень. [13, 14, 15, 16, 17, 18, 61]

Сучасні методи розпізнавання включають:

1) Класифікація зображень. Це метод при якому кожне зображення рослини класифікується в одну з категорій. Наприклад «хворе» або «здорове». Для цієї мети використовують згорткові нейроні мережі (CNN), які ефективно працюють з великим обсягом зображень і дозволяють точно класифікувати об’єкти за різними ознаками.
2) Детекція об’єктів. Включає в себе не лише визначення стану рослини, а й локалізацію пошкоджених частин, наприклад, пошкоджених листів. Для цієї мети застосовують такі архітектури, як YOLO (You Only Look Once) та Faster R-CNN, які дозволяють здійснити детекцію об’єктів на зображенні в реальному часі з високою точністю.
3) Сегментація зображень. Метод збільшує точність визначення меж об’єктів на зображеннях, наприклад, при відокремленні листя рослин від фону. Використовують такі методи як U-Net або Mask R-CNN які спеціалізуються на точному виділенні контурів об’єктів. Цей підхід є важливим для подальшого аналізу, оскільки дозволяє краще розуміти локалізацію проблем.

1.2 Опис вхідних даних дослідженя

1.2.1 Опис вхідних даних дослідження

Вхідні дані дослідження є основою для використання нейронних мереж для автоматизованого аналізу стану рослин. Для цього використовуються зображення різних форматів, що містять знімки рослин в різних умовах, а також зображення, отримані за допомогою різних джерел. Оскільки мета дослідження

полягає в аналізі зображень рослин для їх діагностики, дані мають певні характеристики та виклики при їх обробці.

Формати вхідних даних вже були описані у пункті 1.1.1, далі розглянемо структуру кожного з них. [19, 20, 22, 23]

У RGB кожне зображення представлено у вигляді матриці розміром (h,w,3), де h - висота зображення, w - ширина зображення, а 3 - кількість кольорових каналів. Зображення може бути різних розмірів але найчастіше використовуються зображення з розподільною здатністю від 256x256 до 1024x1024 пікселів.

Обробка RGB зображень вимагає значних обчислювальних ресурсів, оскільки кожен піксель зображення має три значення (для кожного каналу). Це збільшує обсяг даних, що потребують більшої пам'яті для зберігання і більших потужностей для обробки.

Монохромні зображення структурно представлено матрицею розміром (h,w,1), де h - висота зображення, w - ширина зображення, а 1 - єдиний канал, що є показником інтенсивності світла.

Втрата кольорової інформації обмежує можливості діагностики, особливо в тих випадках, коли важливі кольорові зміни (наприклад, у разі пошкодження листя). Однак монохромні зображення зменшують навантаження на обчислювальні ресурси, що дозволяє зберігати і обробляти більшу кількість даних.

ІЧ зображення структурно представлено матрицею розміром (h,w,1), де h - висота зображення, w - ширина зображення, а 1 - єдиний канал, що є показником температури або іншого фізичного параметру що визначаєтьсь через інфрачервоне випромінювання.

ІЧ зображення часто мають меншу роздільну здатність порівняно з RGB зображеннями, і часто вони потребують додаткової обробки для усунення шумів або артефактів, що виникають через інші фактори, як вологість, погодні умови або невірні налаштування сенсорів. Окрім того, їх важко інтерпретувати без відповідних знань фізичних властивостей рослин та навколишнього середовища.

1.2.2 Структура та організація вхідних даних

Дані зображень зберігаються у відповідних форматах (jpg, png, tif, bmp та інші) і організовані в директорії відповідно до класів рослин або їх стану (здорові, хворі, пошкоджені). Для цього використовуються структури директорій, в яких кожна піддиректорія відповідає певному класу чи хворобі рослини.

Дані розподіляються на три основні набори - тренувальний, валідаційний та тестовий. Це дає змогу моделі навчатися на різних частинах набору даних та перевіряти свою здатність правильно класифікувати нові, раніше невідомі зображення.

Тренувальний набір містить найбільшу кількість зображень і використовується для навчання моделі.

Валідаційний набір призначений для налаштування гіперпараметрів моделі та визначення її ефективності під час навчання.

Тестовий набір використовується для остаточної перевірки точності моделі після її тренування.

Окрім зображень, часто присутні анотації, які допомагають точно вказати місце захворювання та/або пошкодження на зображеннях. Формати анотацій можуть бути різними залежно від використаної моделі.

YOLO формат це зберігання інформації про координати обмежувальних рамок (bounding boxes) і класи об’єктів у текстовому файлі.

Pascal VOC забезпечує біль детальну розмітку та має багато додаткових параметрів. [19]

Анотації допомагають алгоритмам точніше розпізнавати об’єкти на зображенні та класифікувати їх по категоріям.

1.3 Аналіз існуючих технологій розпізнавання та класифікації рослин

1.3.1 Аналіз існуючих методів дослідження

Вивчення методів аналізу зображень рослин та діагностики з використанням нейронних мереж є ключовим етапом для розробки ефективних систем автоматизованого моніторингу стану рослин. В останні роки з`явилося , багато рішень, заснованих на застосуванні комп'ютерного зору та машинного навчання, що дозволяють автоматично класифікувати стан рослин, виявляти хвороби і навіть пропонувати рекомендації для їх лікування. Це включає як загальні методи класифікації зображень, так і спеціалізовані рішення для окремих умов, таких як гідропоніка або використання супутникових даних для глобального моніторингу.

Існує ряд готових програмних рішень, які базуються на методах машинного навчання та використовують різні типи зображень для класифікації рослин. Нижче наведено кілька прикладів таких рішень:

1) PlantVillage це відкрите програмне забезпечення, що використовує нейронні мережі для класифікації рослин. PlantVillage містить велику базу зображень рослин з різними здоровими рослинами та їх захворюваннями. Модель на основі CNN навчається на цих зображеннях і може класифікувати нові зображення, визначаючи, чи є на них хвороби. [24]
2) DeepGreen - програмне забезпечення для автоматизованої діагностики рослин у теплицях та агропромислових комплексах яке використовує глибоке навчання для виявлення хвороб на основі зображень з камер. Система здатна обробляти великі обсяги даних та надавати рекомендації по лікуванню рослин. [26, 27]
3) Agri-Tech Solutions - це програмне забезпечення для обробки супутникових зображень з визначення стану сільськогосподарських культур на основі аналізу зображень, що були отримані з космосу. Це рішення допомагає великим компаніям здійснювати моніторинг стану

рослин на великих площах, що є важливим для прийняття рішень на глобальному рівні.

1.3.2 Космічні рішення

Використання супутникових зображень для моніторингу рослинності є важливим інструментом для аналізу великих територій, таких як сільськогосподарські угіддя або лісові масиви. Завдяки супутниковим даним можна здійснювати оцінку стану рослин на великих площах, виявляючи можливі проблеми на ранніх стадіях.

Основні приклади таких рішень:

1) Sentinel-2 - супутники європейської програми Copernicus, які здатні надавати високоякісні зображення з великою роздільною здатністю. Використовуючи ці зображення, можливо здійснювати моніторинг стану рослин та виявляти їх стресові стани та хвороби. [30, 31]
2) Landsat - супутникові зображення від NASA, що використовуються для вивчення змін земного покрову, в тому числі для сільськогосподарських культур. Дані від Landsat дозволяють проводити довгостроковий моніторинг і спостерігати за змінами на великій території. [32, 33]

1.3.3 Рішення для гідрапоніки

Унікальність вирощування рослин у гідропонних системах полягає в тому, що рослини вирощуються без грунту, що вимагає специфічного підходу до їх моніторингу та діагностики. В цих системах критично важливо вчасно виявляти зміни в стані рослин, оскільки вони можуть бути дуже чутливі до змін у навколишньому середовищі, таких як температура води, рівень pH та концентрація поживних речовин.

Приклад такої системи:

1) Hydroponic Visual Monitoring System - це система яка використовує камери для постійного моніторингу стану рослин у гідропонних установках. Система використовує алгоритми класифікації та

сегментації для виявлення змін у рослинах, таких як пожовтіння листя або ознаки захворювань. [34, 35]

1.4 Постановка завдання

У результаті аналізу існуючих методів дослідження та вивчення застосування нейронних мереж для класифікації та діагностики стану рослин, можна сформулювати основне завдання, яке є метою даної магістерської роботи. Завдання полягає в розробці програмного забезпечення, яке автоматизує процес моніторингу здоров'я рослин, виявляючи хвороби. Для досягнення цієї мети необхідно виконати наступні етапи.

Створення набору даних. Першим етапом є створення повноцінного та різноманітного набору даних для навчання нейронної мережі. Це включає збір зображень рослин з різними захворюваннями та в різних умовах, анотації до об’єктів, розподіл даних на тренувальні, валідаційні та тестові набори.

Розробка інформаційної системи для автоматизованої діагностики. Після підготовки даних необхідно спроектувати та створити програмне забезпечення, яке реалізує алгоритми машинного навчання для аналізу зображень рослин. Основними етапами є створення діаграм, що описують систему, розробка нейронної мережі, яка зможе класифікувати стан рослин, а також локалізувати області, інтеграція інтерфейсу для взяємодії з користувачем.

Завершальним етапом є апробація розробленої системи, що включає у себе тестування на нових, раніше невідомих зображеннях, оцінка точності та ефективності на основі метрик, таких як F1-міра та інші та оцінка можливості використання рограмного забезпечення у реальних умовах аграрного сектору та гідропоніки.

Основне завдання полягає у створенні ефективної та універсальної системи, здатної автоматично аналізувати стан рослин і допомагати фермерам або садівникам виявляти та лікувати хвороби рослин, зменшуючи витрати на моніторинг та збільшуючи врожайність сільськогосподарських культур.

1.5 Висновок

У результаті виконаних досліджень та аналізу сучасних методів класифікації та діагностики стану рослин, можна зробити кілька ключових висновків.

Використання нейронних мереж, зокрема згорткових (CNN) і гібридних моделей, для автоматизованої класифікації стану рослин є надзвичайно перспективним напрямком. Ці методи дозволяють не лише ефективно виявляти хвороби рослин, але й прогнозувати їх розвиток на основі аналізу зображень різних типів, таких як RGB, монохромні та інфрачервоні зображення.

Одним із ключових етапів розробки є створення якісного набору даних для тренування моделей. У процесі роботи над дослідженням визначено, що наявність великої кількості різноманітних зображень рослин, правильно анотованих та збалансованих, є основою для побудови ефективних алгоритмів. Важливим є також розподіл даних на тренувальні, валідаційні та тестові набори для забезпечення надійності і точності моделі.

Розробка інформаційної системи для автоматизованої діагностики стану рослин є важливим кроком до інтеграції інноваційних технологій в сільське господарство. Встановлення чітких алгоритмів класифікації та локалізації пошкоджених областей рослин може значно знизити витрати на моніторинг та збільшити врожайність.

Після створення інформаційної системи важливою частиною є його апробація. Результати тестування на реальних даних показують, що застосування методів машинного навчання для моніторингу стану рослин забезпечує високу точність і ефективність виявлення захворювань. Проте для досягнення найкращих результатів необхідно враховувати різноманітність умов зйомки та покращувати точність локалізації пошкоджених ділянок.

Використання зображень для моніторингу стану рослин для гідропоніки, відкриває нові можливості для точного моніторингу стану рослин в реальному часі. В умовах гідропоніки, де рослини чутливі до змін у навколишньому

середовищі, ці системи дозволяють отримувати швидкі та точні дані для прийняття рішень щодо оптимізації умов вирощування.

Подальший розвиток та вдосконалення технологій комп`ютерного зору та нейронних мереж має величезний потенціал для автоматизації процесів діагностики та моніторингу стану рослин. Впровадження таких рішень у сільське господарство, зокрема в гідропоніці та агропромислових комплексах, може суттєво підвищити ефективність вирощування рослин та зменшити витрати на моніторинг.

РОЗДІЛ 2 ВИКОРИСТАНІ МАТЕМАТИЧНІ МОДЕЛІ

2.1 Структура даних

Для успішної розробки системи автоматизованої діагностики стану рослин на основі зображень необхідно чітко розуміти ієрархію даних та структуру, в якій зберігаються інформація про кожну рослину. Вона включає в себе різні рівні абстракції, від базових пікселів до складних об'єктів, таких як листя, стебла або окремі захворювання. Вся ця інформація є частиною загальної матриці даних, яку використовує нейронна мережа для прийняття рішень.

2.1.1 Пікселі

На найнижчому рівні дані представлені пікселями які є найменшими одиницями зображення. Кожен піксель містить значення, що відповідає кольору або інтенсивності світла в точці зображення. [36, 37]

Для RGB зображення кожен піксель має три значення, що відповідають каналам кольору (червоний, зелений, синій). Це значення можуть бути в межах від 0 до 255 для кожного каналу, де 0 означає відсутність кольору, а 255 — максимальну інтенсивність кольору.

Для прикладу піксель з координатами (x,y) в зображені може мати таке значення: (120,150,200) де для червоного каналу значення буде дорівнюватися 120, для синього каналу буде дорівнюватися 150, а для зеленого каналу буде дорівнюватися 200.

Для монохромних зображень кожен піксель містить лише одне значення, яке визначає інтенсивність світла в даній точці зображення, що є корисним для задач, де кольори не є основними ознаками.

Пікселі інфрачервоних зображень містять інформацію про температуру або інші фізичні параметри рослин на основі інфрачервоного випромінювання. Це значення може бути в межах від 0 до 255 або мати інші діапазони залежно від сенсорів, що використовуються для збору даних.

2.1.2 Об’єкти

Наступний рівень у структурі даних - це об’єкти. В рамках дослідження стану рослин, об’єктами можуть бути окремі частини рослин, такі як листя, стебла, гілки або навіть окремі хвороби, що виявляються на зображеннях.

На цьому рівні система повинна визначати, чи є об'єкт на зображенні (наприклад, здоровий листок, хворий листок, стебло або квітка) та яку категорію цей об’єкт належить. Наприклад, для завдання класифікації зображень рослин, нейронна мережа має навчитися розпізнавати різні стани рослин. Для цього використовуються обмежувальні рамки (bounding boxes), що вказують на розташування об’єкта в координатах зображення. Наприклад об’єкт «хворий листок» буде виявлений на зображенні і позначений прямокутною рамкою. [38, 39]

2.1.3 Матриці

Усі пікселі на зображенні формують матрицю даних, яка є основною структурою, що використовується нейронною мережею для обробки інформації.

Для зображень у форматі RGB матриця виглядає як тривимірний масив даних, що складається з висоти, ширини та трьох кольорових каналів.

Для монохромних зображень матриця буде двовимірною, де кожен елемент матриці відповідає інтенсивності одного пікселя зображення.

Інфрачервоні зображення також можуть бути представлені у вигляді двовимірних або тривимірних масивів, залежно від кількості використовуваних параметрів. [40, 41]

2.1.4 Ієрархія даних в задачах комп’ютерного зору

У задачах комп'ютерного зору для моніторингу стану рослин можна виділити кілька рівнів ієрархії даних. [42, 43]

На нижчому рівні пікселі, які складають базову інформацію про колір, інтенсивність або температуру.

На середньому рівні об’єкти, що відповідають конкретним частинам рослини або певним явищам (захворювання, пошкодження).

На вищому рівні матриці які об’єднують всю інформацію про пікселі і об’єкти, створюючи повне зображення, яке система у подальшму використовує для прийняття рішень

2.2 Методи та моделі

2.2.1 Згорткові мережі

Згорткові нейронні мережі (CNN, Convolutional Neural Networks) - це тип глибоких нейронних мереж, призначений для обробки даних із сітчастою структурою, таких як зображення. Вони ґрунтуються на принципі згортки, що дозволяє ефективно виділяти просторові та ієрархічні ознаки зображень. Розглянемо докладно кожну з найпопулярніших архітектур CNN. [16, 17]

LeNet яку розробив Ян Лекун (Yann LeCun) у 1989-у році з ціллю розпізнавання рукописних цифр. LeNet складається з згорткових і підвибіркових (subsampling) шарів, що чергуються, за якими слідують повнозв’язні шари. Ця архітектура відносно проста:

1) вхідний шар: приймає зображення 28×28;
2) два згорткові шари з функцією активації Tanh;
3) два шари пулінгу (average pooling);
4) повнозв’язні шари для класифікації.

Це перша успішна CNN що довела ефективність згорткових нейронних мереж. Використовується здебільшого для вирішення базових завдань з класифікації.

AlexNet що була створена Алексом Крижевським (Alex Krizhevsky) та його співавторами у 2012-у році для участі в ImageNet Large Scale Visual Recognition Challenge (ILSVRC). Структурно Alex Net має 8 шарів:

1) п’ять згорткових шарів з функцією активації ReLU;
2) три повнозв’язних шари у кінці.

Використовується макспулінг (MaxPooling) для зменшення розмірності

ознак. З особливостей цієї архітектури виділяється те, що для збільшення швидкості навчання використовується GPU. Також замість функцій активації Sigmoid\Tanh було використано ReLU. AlexNet мала революційне значення бо значно перевершила усі попередні методи по своїй точності та поклала початок епосі глибокого навчання. [13]

VGGNet розроблена візуальною геометричною групою (Visual Geometry Group) оксфордського університету у 2014-у році для класифікації та локалізації об’єктів. VGGNet представлена декількома версіями (VGG-16, VGG-19) де числа вказують на кількість шарів. Використовує лише 3x3 згорткові шари фільтрації для спрощення структури:

1) глибокі архітектури з послідовними зготковими шарами (до 19-и);
2) макспулінг (MaxPooling) для зменшення розмірності;
3) декілька повнозв’язних шарів у кінці і SoftMax.

З особливостей це проста та зрозуміла архітектура з великою кількістю шарів, що підходить для детальної класифікації недоліком якої є великі обчислювальні витрати через глибину мережі.

ResNet (Residual Networks) яка була розроблена командою Microsoft у 2015-у році. Ціллю створення цієї архітектури було вирішення проблеми затухання градієнта при дуже глибокому навчанні. ResNet представляє собою мережу з залишковими блоками (residual blocks) де кожен блок додає вхідні дані (input) до виходу (output) створюючи так звані пропуски (skip connnections) глибина яких може досягати 152-х шарів (ResNet-152). Особливістю ResNet є використання пропусків (skip connections) що дозволяє передавати інформацію через декілька шарів уникаючи втрати важливої інформації та покращуючи навчання. Ця архітектура зменшує перенавчання навіть для дуже глибоких мереж і має велику ефективність у задачах з класифікації та детекції об’єктів. [45]

Порівняння CNN архітектур зображено на таблиці 2.1..

Архітектура	Рік	Застосування	Переваги
LeNet	1989	Прості задачі з класифікації (наприклад числа)	Легка та ефективна для базових завдань
AlexNet	2012	Класифікація ImageNet	Швидке навчання на GPU, використання ReLU
VGGNet	2014	Детальна класифікація	Проста архітектура, висока точність
ResNet	2015	Глибокі моделі для класифікації та детекції	Успішне навчання надглибоких мереж

Таблиця 2.1. - Порівняння CNN архітектур.

2.2.2 Об’єктні детектори

Об’єктні детектори - це моделі, які не тільки класифікують об’єкти на зображенні, але й визначають їхнє розташування за допомогою прямокутних рамок (bounding boxes). Застосування об'єктних детекторів: автономні системи керування (автомобілі, дрони), системи відеоспостереження та безпеки, аналіз зображень у медицині (наприклад, виявлення пухлин), агросектор (наприклад, виявлення пошкоджених рослин). Розглянемо основні підходи до детекції об’єктів.

Faster R-CNN була розроблена командою Microsoft у 2015-у році з ціллю детекції об’єктів з високою точністю. Архітектура будується на основі трьох основних компонентів:

1) згорткова нейронна мережа (backbone) що використовується для отримання ознак вихідного зображення;
2) регіональна мережа пропозицій RPN (Region Proposal Network) що визначає потенційні області де можуть бути об’єкти;
3) класифікатор який приймає області, запропоновані RPN, уточнює їх координати та класифікує об’єкт.

Принцип роботи Faster R-CNN: на першому етапі RPN генерує регіони-кандидати (області де ймовірно знаходяться об’єкти) далі ці регіони уточнюються і класифікуються. Перевагами Faster R-CNN є висока точність і здатність працювати з невеликими об’єктами та універсальність для різноманітних типів зображень. Недоліки цієї архітектури це відносно велика обчислювальна складність та мала швидкість у порівнянні з новітніми підходами, наприклад YOLO. [15]

YOLO (You Only Look Once) була розроблена Джозефом Редмоном (Joseph Redmon) і його співавторами у 2016-му році для реалізації швидкої детекції в реальному часі. YOLO перетворює задачу детекції у задачу регресії. Зображення розбивається на сітку (наприклад 7x7) де кожна клітка прогнозує декілька bounding boxes і ймовірність класів. Всі об’єкти обробляються одночасно, що забезпечує високу швидкість. Принцип роботи полягає у тому, що модель дивиться лише один раз (від чого і отримала свою назву) на зображення і прогнозує одразу усі bounding boxes після чого використовуються anchor boxes для прогнозування об’єктів різного розміру. Перевагами цієї моделі є дуже велика швидкість, що підходить для задач реального часу та простота архітектури. Недоліком є менша точність ніж Faster R-CNN для задач, що потребують високої деталізації. [14, 61]

SSD (Single Shot MultiBox Detector) розроблена Google у 2016-у як модель що балансує між точністю та швидкістю. SSD має наступну архітектуру:

1) згорткова нейронна мережа, розширена для прогнозування bounding boxes на різних рівнях;
2) пірамідальні слої у яких ознаки беруться на різних масштабах, що дозволяє працювати з об’єктами різних розмірів;
3) для кожного з рівнів bounding boxes і класи.

Недоліком є те, шо SSD програє Faster R-CNN в задачах де точність є пріоритетом, також SSD потребує великої кількості даних для тренування. [46]

Порівняння об’єктні детекторів зображено на таблиці 2.2..

Архітектура	Рік	Застосування	Переваги
Faster R-CNN	2015	Виявлення об'єктів, завдання з високою точністю	Висока точність, здатність працювати з дрібними об’єктами.
YOLO	2016	Задачі у реальному часі	Висока швидкість, простота архітектури
SSD	2016	Загальні задачі детекції об’єктів	Баланс швидкості і точності

Таблиця 2.2. - Порівняння об’єктних детекторів.

2.2.3 Сегментаційні нейронні мережі

Сегментація зображень - це завдання, в якому необхідно розділити зображення на кілька частин (сегментів), кожна з яких відповідатиме певному об'єкту або області інтересу. У завданнях сегментації важливо як визначити присутність об'єкта, а й точно вказати його форму, контури і розташування. Сегментаційні нейронні мережі, як правило, використовуються в медицині, агрономії, робототехніці та інших галузях, де потрібне точне виділення об'єктів на зображенні. Розглянемо найпопулярніші архітектури сегментації.

U-Net розроблена у 2015-у році науковцями з Університету Фраунгофера (Fraunhofer-Institut für Integrierte Schaltungen) Германія в першу чергу для медичної сегментації. U-Net складається з двох основних частин:

1) енкодер - стискаюча частина мережі, яка послідовно витягує ознаки зображення, використовуючи згортки та операції пулінгу;
2) декодер - розширювальна частина, яка відновлює просторові роздільні здатності зображення з використанням операцій транспонованих згорток (або upsampling).

Важливою особливістю U-Net є наявність пропусків (skip connections) між шарами енкодера і декодера, що дозволяє зберегти більш точну інформацію про просторові ознаки. Це особливо важливо для точної сегментації невеликих об’єктів. Кожне з’єднання в U-Net поєднує виходи з шарів енкодера з входами у відповідний шар декодеру покращуючи реконструкцію зображень. Таким чином мережа може біль точно сегментувати об’єкти на зображенні зберігаючи контури і деталі, що особливо важливо у медицині для сегментації опухолей та клітин. Недоліком U-Net є те, що можуть виникати проблеми при обробці зображень високої роздільної здатності, а також потребують значних обчислювальних ресурсів під час навчання великих датасетах. U-Net використовується у медичному напрямку для розподілу опухолей та здорових тканин, наприклад при МРТ, а також для геоінформаційних систем (сегментація супутникових знімків). [47]

Mask R-CNN розроблена у 2017-у році командою Facebook AI Research (FAIR) як розширення методу Faster R-CNN для сегментації об’єктів. Архітектура Mask R-CNN складається з таких блоків:

1) основний блок - Mask R-CNN використовує RPN (Region Proposal Network) для генерації кандитатів на об’єкти (як це було в Faster R-CNN);
2) новий блок Mask - для кожного з пропонованних об’єктів модель генерує не тільки його положення (bounding boxes) та клас, а також і сегментаційну маску, маска представляє собою бінарне зображення де кожен піксель вказує на приналежність його до об’єкту чи ні;
3) маски генеруються з використанням окремої мережі для сегментації яка навчається паралельно з основним процесом детекції об’єктів.

Mask R-CNN вирішує завдання детекції та сегментації об'єктів одночасно. Для кожного кандидата (об’єкта), визначеного за допомогою RPN модель генерує bounding box, класифікує об’єкт і створює маску, яка точно виділяє його контури. На відміну від U-Net, Mask R-CNN працює з кількома об’єктами на зображенні і може виконувати завдання детекції, так і сегментацію одночасно.

Недоліками Mask R-CNN є висока обчислювальна складність, особливо у великих зображеннях з безліччю об'єктів а також вона може вимагати більше даних для навчання порівняно з U-Net, оскільки мережа має працювати з кількома об'єктами та масками одночасно. [48]

Порівняння сегментаційних нейронних мереж зображено на таблиці 2.3..

Архітектура	Рік	Застосування	Переваги
U-Net	2015	Сегментація об’єктів	Висока точність при малому наборі даних, простота
Mask R-CNN	2017	Детекція і сегментація об’єктів	Детекція та сегментація об'єктів в одному процесі

Таблиця 2.3. - Порівняння сегментаційних нейронних мереж.

2.2.4 Трансформери для обробки зображень

Трансформери спочатку були розроблені для обробки послідовних даних, таких як текст, у рамках завдання машинного перекладу та інших завдань у галузі природної мови. Однак їхній успіх у обробці текстових даних надихнув на адаптацію механізму уваги (attention mechanism) для обробки зображень. Основна перевага трансформерів полягає в їхній здатності ефективно обробляти довгострокові залежності між елементами вхідних даних, що дозволяє моделям враховувати контекст на вищому рівні. Застосування трансформерів для зображень стало особливо актуальним із розвитком великих обсягів даних та потужних обчислювальних ресурсів.

Vision Transformer (ViT) був представлений у 2020-у році дослідниками Google Research. Ключові особливості ViT:

1) Vision Transformer (ViT) використовує архітектуру трансформера, адаптовану для роботи із зображеннями. На відміну від нейронних згорткових мереж (CNN), які обробляють зображення через локальні

фільтри (згортки), ViT розбиває зображення на невеликі патчі фіксованого розміру, а потім представляється їх як послідовність елементів (як слова в тексті). Ці патчі (наприклад, розміром 16x16 пікселів) лінійно перетворюються на векторне уявлення і передаються модель трансформера. На відміну від традиційних CNN, де пакунки захоплюють локальні залежності, трансформери ViT можуть захоплювати глобальні залежності на зображенні, що робить їх потужними для роботи з складнішими структурами даних;

2) у ViT застосовується механізм self-attention, який дозволяє моделі фокусуватися на різних частинах зображення для отримання інформації про контекст і взаємозв'язки між різними об'єктами. Це відрізняється від згорткових нейронних мереж, які зазвичай обмежуються фокусуванням на невеликих локальних областях зображення;
3) для навчання Vision Transformer потрібно багато даних, щоб модель могла ефективно захоплювати глобальні патерни. Для покращення результатів ViT використовує навчання на великих наборах даних, таких як ImageNet, а потім донавчання для конкретних завдань (наприклад, класифікація або детекція об'єктів).

Недоліками ViT є вимоги до даних. Для досягнення хороших результатів ViT потрібна велика кількість навчальних даних. На менших наборах даних трансформери можуть показувати менш задовільні результати проти CNN. А також обчислювальні ресурси. Трансформери, особливо ViT, вимагають значних обчислювальних ресурсів, оскільки вони обробляють зображення як послідовності токенів (патчів), що збільшує час обчислень та використання пам'яті. Проте ViT успішно використовується для класифікації завдань, таких як класифікація об'єктів на зображеннях, де потрібна висока точність і здатність до обробки глобальних залежностей. ViT корисний у завданнях, які потребують розпізнавання складних структур та об'єктів, таких як медичні зображення, супутникові знімки, аналіз зображень з кількома об'єктами. Цей підхід є

особливо добрим для обробки даних з високою роздільною здатністю та великими розмірами зображень. [49]

Swin Transformer розроблений у 2021-у році був представлений Swin Transformer (Shifted Window Transformer), який є покращенням для Vision Transformer та вирішує проблему роботи із зображеннями різної розподільної здатності. На відміну від ViT, Swin Transformer використовує концепцію зсувів вікон (shifted windows) для ефективнішого захоплення локальних та глобальних залежностей у зображенні. Вікна (windows) фіксованого розміру застосовуються до зображення для отримання локальних ознак, і ці вікна зсуваються між шарами, що дозволяє захоплювати інформацію з різних рівнів зображення. На нижчих рівнях моделі використовуються локальні вікна, щоб ефективно отримувати дрібні ознаки. На вищих рівнях за допомогою зсуву вікон модель може захоплювати глобальні залежності та вивчати складніші зв'язки на зображенні. Swin Transformer легше масштабується для зображень різного розміру та працює швидше у порівнянні з ViT. Він також демонструє відмінні результати на різних завданнях комп'ютерного зору, включаючи класифікацію, детекцію та сегментацію. [50]

Трансформери, такі як ViT та Swin Transformer, демонструють високий потенціал у галузі обробки зображень. Їхня здатність захоплювати глобальні залежності та обробляти великі обсяги даних робить їх вкрай ефективними для складних завдань у комп'ютерному зорі. Хоча вони вимагають значних обчислювальних ресурсів та великого обсягу даних для навчання, їх результати у завданнях класифікації та детекції об'єктів показують перевагу над традиційними підходами до великих даних.

2.2.5 Гібридні моделі

Гібридні моделі - це нейронні мережі, які поєднують різні архітектури або підходи для вирішення завдань, де необхідно використовувати переваги декількох типів мереж. Комбінування різних моделей дозволяє створити більш потужні та універсальні рішення, які можуть ефективно справлятися з різними

аспектами даних, таких як часові залежності, просторові ознаки та багатозадачність. [38, 39]

Прикладом гібридних моделей є CNN + RNN, CNN + Transformer, CNN + GAN та інші. Ці моделі використовуються, коли завдання вимагає поєднання ознак, витягнутих із зображень, з аналізом тимчасових або послідовних даних, що дозволяє моделі ефективніше сприймати інформацію та приймати рішення.

Комбіновані моделі CNN і RNN часто використовуються для завдань, де потрібно враховувати як просторові, так і тимчасові залежності. Наприклад, для аналізу відеопотоку, прогнозування часових рядів або роботи з послідовностями зображень. CNN добре справляються із вилученням просторових ознак із зображень, у той час як RNN (особливо LSTM і GRU) здатні захоплювати тимчасові залежності та передбачати майбутні події на основі попередніх.

CNN + Transformer (Згорткові нейронні мережі + Трансформери) тип моделі який комбінує потужність згорткових мереж для отримання локальних ознак із зображень з ефективністю трансформерів у захопленні глобальних залежностей. Це дозволяє вирішувати завдання, що вимагають точності в обробці зображень, так і здатності вловлювати довгострокові зв'язки між об'єктами або елементами зображення.

Гібридні моделі CNN + GAN використовуються для завдань, де необхідно не тільки отримувати ознаки із зображень, але й генерувати нові зображення або дані, схожі на вихідні. Генеративні змагальні мережі (GAN) добре підходять для створення зображень, покращення якості або створення нових варіантів даних на основі існуючих.

Гібридні моделі CNN + MLP часто застосовуються, коли потрібно поєднувати витяг ознак з зображень (за допомогою CNN) та їх наступну класифікацію або регресію (за допомогою MLP).

Гібридні нейронні мережі пропонують потужні рішення для завдань, що вимагають використання декількох типів даних або отримання різних типів ознак.

Вони дозволяють поєднувати переваги різних моделей та підходів, що

може призвести до значного покращення точності та ефективності вирішення завдань у галузі комп'ютерного зору, обробки часу та інших областей.

2.3 Висновок

Найважливішим аспектом машинного зору є структура даних, яка є набором зображень, міток та анотацій, необхідних для навчання моделей. У контексті зображень дані можуть бути представлені як пікселі, які, в свою чергу, утворюють матриці значень, що відображають різні характеристики об'єкта (наприклад, колір, інтенсивність, текстури). Використання даних з правильною структурою допомагає моделям машинного зору ефективно навчатися та працювати з різними завданнями.

Пікселі зображень служать основою для роботи всіх алгоритмів машинного зору. Кожне зображення представлене як матриця де кожен елемент містить значення інтенсивності кольору. Коли пікселі об`єднуються в о’бєкти, вони стають осмисленими елементами для подальшої обробки моделлю. Перетворення цих даних в о’єктні форми, такі як bounding box або маски сегментації, є критично важливим для задач, таких як класифікація чи детекція об’єктів.

У задачах машинного зору існує кілька рівнів ієрархії, починаючи від низькорівневих ознак, таких як пікселі, до більш високорівневих ознак, таких як цілі об’єкти, сцени та дії. Ефективне використання ієрархії даних дозволяє моделям краще розуміти зображення та робити більш точні передбачення.

Порівняння різних моделей дозволяє вибрати найбільш підходящий метод для вирішення задачі. Моделі машинного зору та їх підходи надають широкий вибір інструментів для вирішення різних задач. Вибір методу залежить від типу задачі (класифікація, детекція, сегментація), а також від вимог до точності, швидкості та складності моделі. Поєднання різних підходів у гібридні моделі є перспективним напрямком для підвищення точності в складних задачах машинного зору.

РОЗДІЛ 3 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ТЕХНОЛОГІЙ

3.1 Огляд програмних платформ та їх функціональних можливостей

У цьому пункті буде розглянуто список вибраних програмних платформ, що використовуються для реалізації цього проекту з коротким описом кожної з них.

3.1.1 Огляд мови програмування Python

Python (рис. 3.1) є однією з найращих мов програмування для розробки нейроннии мереж. Його популярність пояснюється поєднанням зручного синтаксису, розвинутої екосистеми наявних бібліотек, та активної спільноти. [51]

Рис. 3.1. - Логотип Python

Головною силою в сфері машинного навчання є велика кількість ключових інструментів, таких як TensorFlow, PyTorch, Keras, Scikit-learn та інші. Вони відчиняють двері до світу створення нейронних мереж різного рівня складності.

Переваги Phyton для глибокого навчання. Python став стандартом де-факто в області нейронних мереж завдяки безлічі факторів. Насамперед, це його інтуїтивно зрозумілий синтаксис, який дозволяє зосередитись на задачі, а не на технічних деталях. Його кросплатформність і сумісність з більшістю операційних систем роблять його зручним для розробки та запуску програм, також Важливою перевагою Python є підтримка GPU через такі інструменти як CUDA та TensorRT, що дозволяє значно прискорити навчання моделей. Величезна спільнота розробників гарантує доступ до численних навчальних

матеріалів, форумів та оновлень бібліотек.

Щодо мінусів у контексті роботи, Python не позбавлений недоліків. Одним із головних мінусів є його порівняно низька продуктивність у порівнянні з мовами низького рівня, такими як C++ або Rust. Це особливо помітно під час роботи з великими обсягами даних, якщо не використовувати оптимізовані бібліотеки. Ще однією проблемою є обмеження багатопоточності через глобальне блокування інтерпретатора (GIL). Це може уповільнювати виконання завдань, що вимагають паралельної обробки, хоча цю проблему можна обійти за допомогою мультипроцесингу.

Python – це потужний інструмент для розробки нейронних мереж, який поєднує в собі простоту, гнучкість та багату екосистему бібліотек. Він підходить як для початківців, так і для досвідчених розробників, пропонуючи можливості реалізації проектів будь-якої складності. Незважаючи на свої недоліки, Python залишається лідером у галузі машинного навчання та глибокого навчання, забезпечуючи швидкий доступ до найпередовіших технологій.

3.1.2 Огляд фреймворку TensorFlow

TensorFlow (рис. 3.2.), що був розроблений Google представляє собою потужний фреймворк для створення та навчання мереж машинного навчання. Він підтримує роботу з CPU, GPU та навіть TPU, що робить його ідеальним вибором для задач які потребують високу розрахункову потужність. Також TensorFlow пропонує широкий функціонал для побудови графів розрахунків та автоматизації навчання і оптимізації моделей, інтеграція з TensorBoard дозволяє візуалізувати процес навчання, аналізувати похибки та налагоджувати модель. [52]

Рис. 3.2. - Логотип TensorFlow

3.1.3 Огляд фреймворк PyTorch

PyTorch (рис. 3.3.) є прямим конкурентом TensorFlow і був створений Facebook та виділяється завдяки своїй гнучкості та простоті. Одна із основних особливостей це динамічний граф розрахунків котрий робить побудову складних моделей інтуїтивно зрозумілим. Він активно використовується для досліджень так як надає велику гнучкість у експериментах з архітектурами нейронних мереж. Окрім цього, він підтримує автоматичне диференціювання та надає інструменти для роботи з тензорами на GPU. [53]

Рис. 3.3. - Логотип PyTorch

3.1.4 Огляд бібліотеки Keras

Keras (рис. 3.4.) це високорівневий інтерфейс для полегшення роботи з TensorFlow. Його основна задача полягає у тому щоб зробити процес створення нейронних мереж максимально зрозумілим і простим. Keras надає готові шари, функції активації, оптимізатори та можливість налаштування гіперпараметрів, що робить його ідеальним вибором для тих хто тільки починає відкривати для себе нейронні мережі та тих, хто хоче швидко створити прототип моделі. [54]

Рис. 3.4. - Логотип Keras

3.1.5 Огляд бібліотеки Scikit-learn

Scikit-learn (рис. 3.5.) хоч і не створювався для глибокого навчання, але він став незамінним інструментом для базових завдань машинного навчання. Його можливо використовувати для підготовки даних, проведення классифікацій, регресій та кластерізації. Важливою перевагою є наявний широкий набір функцій для для оцінки якості моделі та їх інтерпритації. [55]

Рис. 3.5. - Логотип Scikit-learn

3.1.6 Огляд бібліотек Matplotlib та Seaborn

Для візуалізацій та аналізу даних Python також може запропонувати такі бібліотеки як Matplotlib або Seaborn (рис. 3.6.). Вони допомагають візуалізувати метрики, процесс навчання та великий спектр інших варіантів графічних звітів. [56, 57]

Рис. 3.6. - Логотипи Matplotlib та Seaborn

3.1.7 Огляд середовища розробки PyCharm

PyCharm (рис. 3.7.) створений компанією JetBrains, став одним із найпопулярніших інтегрованих середовищ розробки (IDE) для Python. Його вибирають як новачки, так і досвідчені розробники завдяки зручному інтерфейсу, широкому функціоналу та підтримці багатьох сучасних технологій. Ця IDE ідеально підходить для вирішення різноманітних завдань - від написання скриптів до розробки великих додатків, включаючи проекти в галузі штучного інтелекту та машинного навчання.

Рис. 3.7. - Логотип PyCharm

PyCharm виділяється своїми можливостями для роботи в галузі машинного навчання та аналізу даних. IDE підтримує Jupyter Notebook що дозволяючи

писати інтерактивний код та потім інтегрувати його до основного проекту. Крім цього, PyCharm надає потужні інструменти для налагодження та профілювання, які є особливо важливими при роботі з моделями глибокого навчання.

Ключовими особливостями PyCharm є його універсальність та багатий функціонал. Це потужний інструмент, що поєднує усе необхідне для розробки. PyCharm дозволяє зосередитись на вирішенні завдань, знімаючи з розробника необхідність вручну налаштовувати середовище розробки.

Однак PyCharm має і свої мінуси. IDE є досить вимогливою до ресурсів, що може викликати проблеми на старих або слабких комп'ютерах. А також новачкам може бути складно одразу освоїти всі можливості PyCharm через його багатий функціонал.

PyCharm - це ідеальний вибір для розробки на Python, особливо якщо ви працюєте над великими проектами або завданнями в галузі машинного навчання. Завдяки зручному інтерфейсу, широкій підтримці бібліотек та потужним інструментам PyCharm допомагає зосередитись на створенні якісного коду, мінімізуючи технічні складності. Незважаючи на деякі недоліки, PyCharm залишається одним із найкращих середовищ розробки для Python, поєднуючи передові технології та зручність використання. [58]

3.1.8 Огляд інструмента LabelImg

LabelImg - інструмент для розмітки даних у комп'ютерному зору (рис. 3.8.). В епоху штучного інтелекту та машинного навчання якісні дані є одним з ключових елементів успішної розробки моделей. Особливо це важливо у завданнях комп’ютерного зору, де анотації зображень відіграють майже вирішальну роль. Одним із найпопулярніших інструментів для ручної розмітки об’єктів є LabelImg. Це проста, але потужна програма, яка дозволяє ефективно аннотувати дані для тренування нейронних мереж.

Рис. 3.8. - Логотип LabelImg

Однією з головних переваг LabelImg є саме його простота та зрозумілий інтерфейс навіть на інтуїтивному рівні. Цей інструмент підтримує два найбільш використовуємих типи анотацій для формування датасетів. Pascal VOD (XML) використовується у багатьох классичних алгоритмах, таких як R-CNN. YOLO (TXT) формат, що спеціально оптимізований для одноіменної архітектури. LabelImg дозволяє швидко перемикатись між зображеннями використовуючи гарячі клавіші, що значно пришвидшує роботу, в цей же час під час розмітки анотацій одразу можливо задавати класи для них.

Не зважаючи на певні обмеження LabelImg залишається незамінним інструментом для створення високоякісних датасетів у завданнях комп'ютерного зору. Цей інструмент дозволяє розробнику не втрачати зайвий час на формування анотацій а замість цього приділити увагу розробці моделей та рішень. [59]

3.2 Підготовка даних та робота з датасетом

3.2.1 Опис датасету Plant Diseases Dataset

Plant Diseases Dataset був зібраний та опублікований Саміром Бхатараєм (Samir Bhattarai). Він є вже аугменованим та налічує 86,867 зображень 15-и видів рослин, у цілому з урахуванням рослин та їх хвороб датасет нараховує 38 різних класів. За замовчуванням цей датасет розділено з співвідношеням 80/20 на

тренувальний набір та валідаційний набір, приклад зображень з цього датасету зображено на рис. 3.9-3.10. [60]

Тренувальний набір нараховує 70,295 зображень у 38-и класах приблизно 1,642-2,022 зображення на один клас.

Валідаційний набір нараховує 17,572 зображень у 38-и класах приблизно 410-505 зображень на один клас.

У своєму початковому вигляді цей датасет не має ніякої розмітки чи анотацій, а лише сортування по різним директоріям де назва директорії і є класом до якого потрібно віднести зображення що всередині. Це звісно не зовсім те, що нам потрібно тому датасет потребує подальшого доопрацювання.

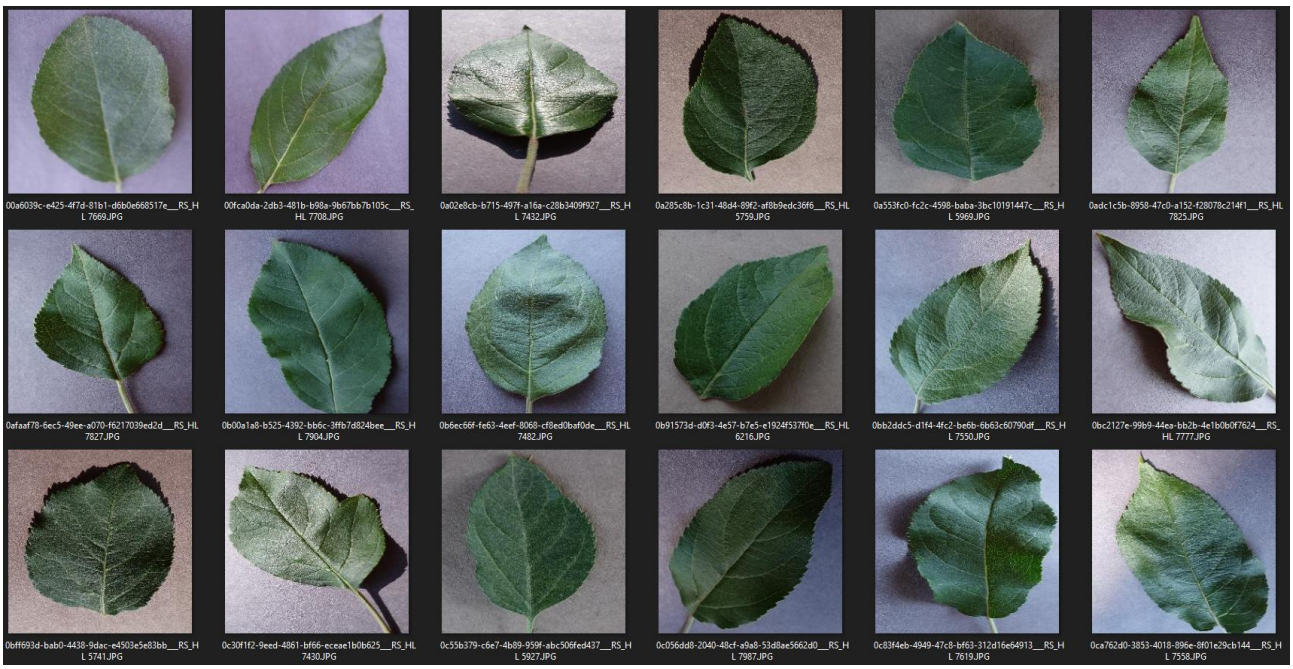

Рис. 3.9. - Зображення з Plant Diseases Dataset що відносяться до класу Apple_healthy

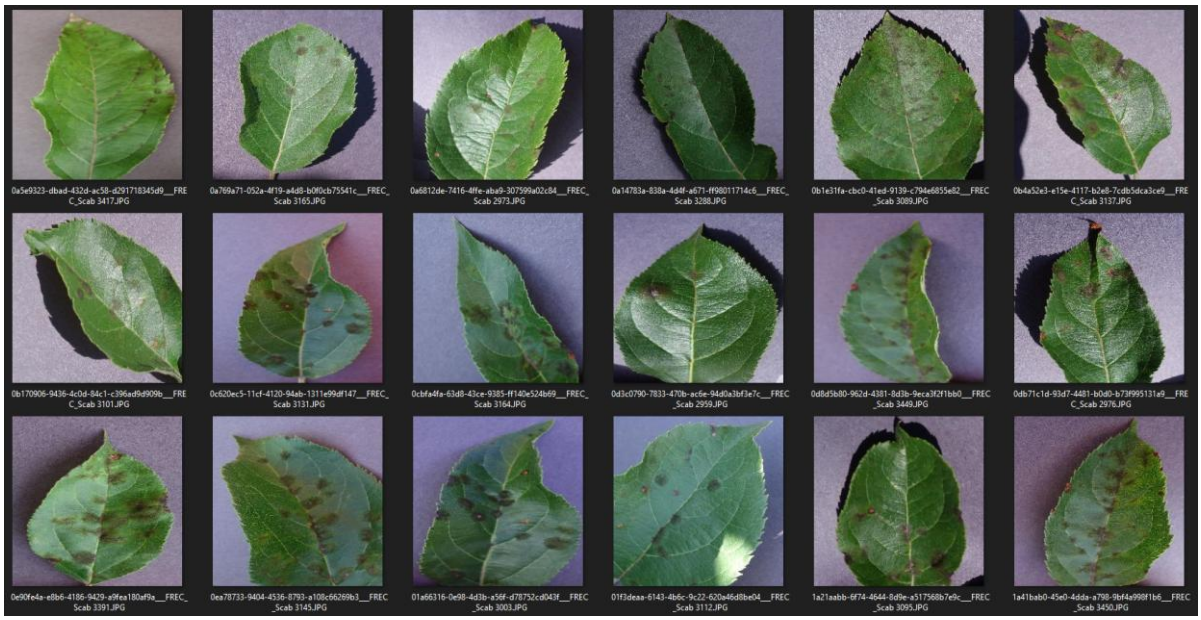

Рис. 3.10. - Зображення з Plant Diseases Dataset що відносяться до класу Apple_scab

3.2.2 Створення файлів анотації та розмітка зображень датасету за допомогою LabelImg

Насамперед перед тим як приступити до розмітки необхідно завантажити LabelImg.

Розробники передбачають багато варіантів так як цей інструмент є кроссплатформним інструкції для його завантаження розглянуті у GitHub репозиторії проекту. Але ми скористаємося останнім зібраним білдом.

Після інсталяції нас зустріне головне вікно цього інструменту (рис. 3.11.), інтерфейс складається з невеликої кількості кнопок, серед яких нам потрібні лише «Open Dir» що дозволяє вибрати директорію у якій ми будемо працювати, за замовчування саме туди будуть надходити створені нами файли анотацій. Також необхідно обрати один з двох варіантів для формату анотацій (Pascal VOC або YOLO). Для переміщення у межах директорії відповідають кнопки «Next Image» та «Prev Image» але перед кожним переходом необхідно зберігати анотацію для файлу в якому ми працювали. Для того що б створити рамку якою виділяється об’єкт класу використовується кнопка «Create\nRectBox».

Усі ці кнопки зображено на рис. 3.12..

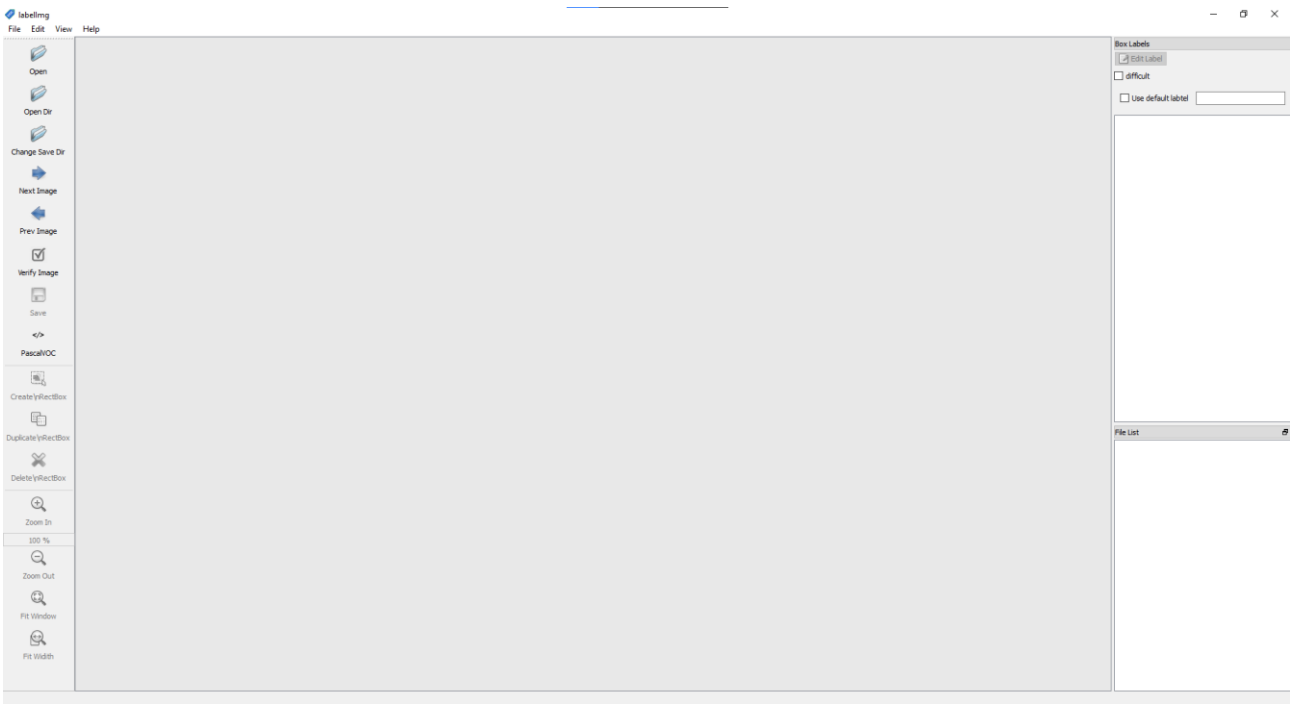

Рис. 3.11. - Головне вікно LabelImg

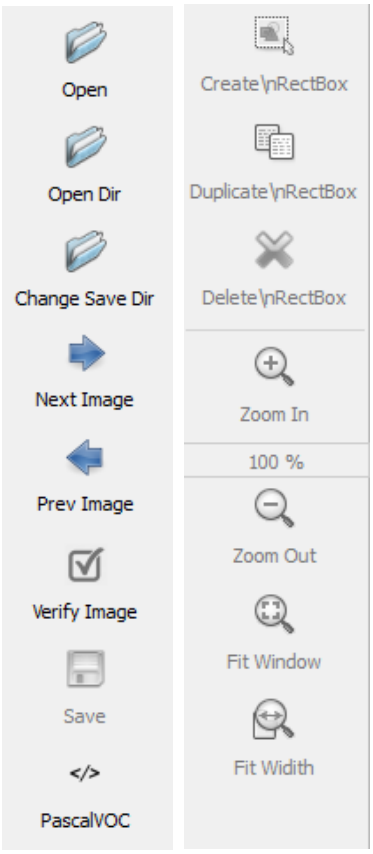

Рис.3.12. - Кнопки через які організоване керування LabelImg

Окрім цього LabelImg має набір гарячих клавіш, шо також можна використовувати для роботи, список цих гарячих клавіш:

1) Ctrl + u - Завантажити всі зображення з каталогу;
2) Ctrl + r Змінити типовий цільовий каталог анотації;
3) Ctrl + s Зберегти;
4) Ctrl + d Копіювати поточну мітку та прямокутник;
5) Ctrl + Shift + d Видалити поточне зображення;
6) Пробіл Позначити поточне зображення як перевірене;
7) w Створіть прямокутник;
8) d Наступне зображення a Попереднє зображення;
9) del Видалити виділене прямокутне поле;
10) Ctrl++ Збільшити;
11) Ctrl-- Зменшити масштаб;
12) ↑→↓← Клавіатурні стрілки для переміщення вибраного прямокутника;

Після того як необхідна директорія відкрита ми побачимо перше зображення яке у ньому розташовано і можемо почати робити анотації. Натиснув «Create\nRectBox» або «w» після чого потрібно мишкою розмітити необхідний об’єкт після чого програма запитає до якого класу слід його віднести (рис. 3.13).

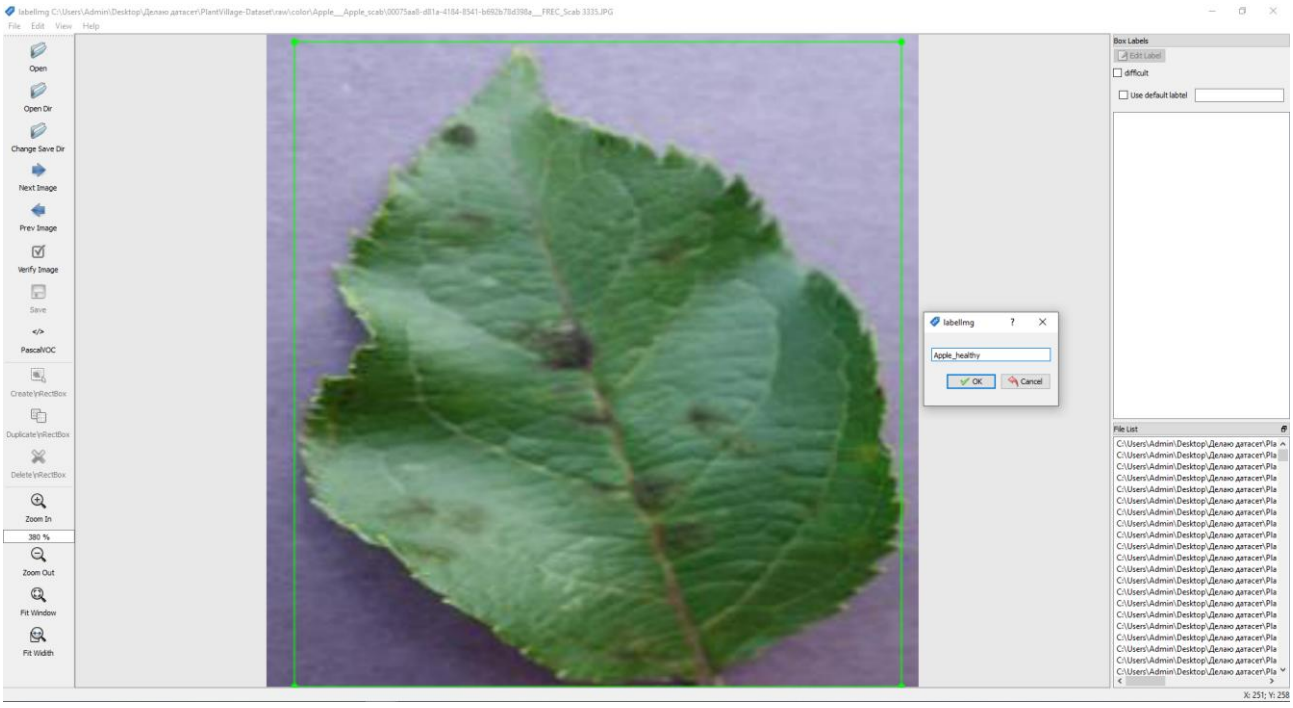

Рис. 3.13. - Створення рамок класу у LabelImg

Далі слід зберегти анотацію та перейти до наступного зображення за допомогою кнопки «Next Image» або «d». Доки усі зображення не будуть розмічені.

Створена анотація матиме наприклад ось такий вид: 1 0.470703 0.498047 0.902344 0.964844. Його можна зрозуміти як <class_id>, <x_center>, <y_center>, <width> <height>. [62]

Class_id є індексом класу, і повинен почитанись з нуля. Для нашого датасету це значення має бути від 0 до 37-и так як ми маємо 38 класів.

X_center, Y_center є координатами центру об’єкта нормалізовані відносно ширини та висоти зображення і можуть бути у діапазоні від 0 до 1 та вираховуються за формулою 3.1 .

$$Xcenter = \frac{Xabs}{widht} \quad Ycenter = \frac{Yabs}{height} \tag{3.1}$$

Де $Xabs$ та $Yabs$ це абсолютні координати центра об’єкта а $widht$ та $height$ - розміри зображення.

Width, height це ширина та висота обмежуючої рамки нормалізовані відносно розміру зображення та вираховуються за формулою 3.2.

$$\text{width} = \frac{box\ width}{(image\ width)} \quad height = \frac{(box\ height)}{(image\ height)} \tag{3.2}$$

Де $box\ width$ та $box\ width$ це розміри обмежуючої рамки у пікселях.

3.2.3 Створення карти імен за допомогою Python скрипта

Для подальшого використання створених нами анотацій необхідно створити мапу імен класів. Це потрібно тому, що нейронна мережа YOLO працює з числовими значеннями для класу, а для читання людиною це не

підходить.

Це можно відносно легко зробити за допомогою скрипта написаного на мові Python і має наступний код:

```
import os

# Path to YOLO directories
yolo_dir = <шлях до директорій>


def generate_class_map(yolo_dir):
    class_map = {}

    # Iterate through directories (each folder represents a class)
    for class_name in sorted(os.listdir(yolo_dir)):
        class_path = os.path.join(yolo_dir, class_name)
        if os.path.isdir(class_path):
            # Find any .txt file in the folder
            txt_files = [f for f in os.listdir(class_path) if
f.endswith('.txt')]
            if not txt_files:
                print(f"Warning: No .txt files found in {class_path}.
Skipping.")
                continue

            # Open the first .txt file found
            txt_path = os.path.join(class_path, txt_files[0])
            with open(txt_path, 'r') as f:
                line = f.readline().strip()

            # Extract the class ID from the first line of the file
            class_id = int(line.split()[0])
            class_map[class_name] = class_id

    return class_map


# Main process
def main():
    class_map = generate_class_map(yolo_dir)

    # Print the class map
    print("class_map = {")
    for class_name, class_id in class_map.items():
        print(f"    '{class_name}': {class_id},")
    print("}")


if __name__ == "__main__":
    main()
```

Принцип роботи цього скрипта дуже простий. Ми скануємо директорію на наявність у ній піддиректорій назву яких ми беремо за назву класу. Відчиняємо цю директорію та читаємо class_id з першого ж txt файлу, що були нами зроблені на етапі створення анотацій. Якщо ж буде знайдена піддиректорія у якій не буде знаходитись жодного txt файлу ми отримаємо відповідне повідомлення з

уточненням у якій саме піддиректорії відсутні txt та пропустить її продовжуючи виконання. Це потрібно що б контролювати правильність виконання коду та у разі проблеми знати з якою саме директорією проблеми.

Як результат роботи скрипту ми отримаємо необхідну нам мапу класів. Це виглядає наступним чином:

```
class_map = {
'class1': 0,
'class2': 1,
'class3': 2,
# ...
}
```

Цю відповідь нам необхідно зберегти бо вона буде дуже важливою у подальшому при навчанні моделі.

3.3 Опис передбаченної моделі та гіперпараметрів

3.3.1 Опис моделі YOLOv8s

Як основу для навчання моделі обрано модель YOLOv8s. Вона представляє собою полегшену версію архітектури і створена для роботи на пристроях з обмеженою потужністю. Вона зберігає баланс між точністю та швидкістю, узагальнена візуалізація архітектури зображена на рис. 3.14. [14, 61, 62]

Модель складається з трьох основних компонентів:

1) Backbone - основна мережа, що отримує ознаки з зображення.
2) Neck - займається посиленням та агрегацією ознак з різними рівнями розподільної здатності.
3) Head - фінальне прогнозування класів та координат об’єктів.

Backbone відповідає за отримання просторових та контекстних ознак, використовуючи модифіковані CSP-блоки (Cross-Stage Partial Network) та працює на трьох рівнях розподільної здатності зображень. Це полегшені блоки оптимізовані для швидкості. Ключовими компонентами якого є Stem, CSP, Bottleneck блоки та Squeeze-and-Excitation (SE).

Stem це початковий згортковий блок який зменшує розмір зображення та збільшує число каналів. Згортка 3x3, BatchNorm, активація Leaky ReLU.

CSP блок складається з двох шляхів. Перший передає ознаки напряму, а другий використовує BottleNeck блоки для обробки ознак. У кінці виконується злиття ознак за допомогою операції concatenation.

Bottleneck блоки складаються з трьох згорток. 1x1 для зменшення розміру, 3x3 для виявлення локальних просторових ознак, 1x1 для відновлення розміру.

Squeeze-and-Excitation (SE) використовується дял виділення найбільш значимих каналів, що посилює увагу моделі на ключових областях зображення.

Neck посилює ознаки, що були отримані з Backbone та агрегує їх. YOLOv8s використовує PANet (Path Aggregation Network), що дозволяє поєднувати інформацію про великі об’єкти с деталями малих об’єктів. Ключовими елементами є FPN (Feature Pyramid network), Bottom-Up Path Augmentation, Fusion блоки.

FPN (Feature Pyramid network) відповідає за потік інформації від нижніх рівнів до верхніх. Це дозволяє враховувати контекст великих об’єктів.

Bottom-UP Path Augmentation навпаки відповідає за потік інформації від верхніх шарів до нижніх. Це дозволяє посилити просторові ознаки.

Fussion блоки використовуються для поєднання ознак через операції складання та множення (concatenation).

Head відповідає за фінальну класифікацію об’єктів та їх координати. YOLOv8s використовує підхід без якорей (anchor-free), що робить процес навчання легшим та підвищує точність. Ключовим элементом э Output Layers. Це згорткові шари для генерації трьох наборів передбачень для трьох масштабів. Ці вихідні параметри складаються з координатів (z, y, w, h) що описують центр об’єкта, ширину та висоту. Класи об’єкта як ймовірність віднесення його до конкретного класу. Та Confidence що відібражає впевненість моделі щодо наявності об’єкта.

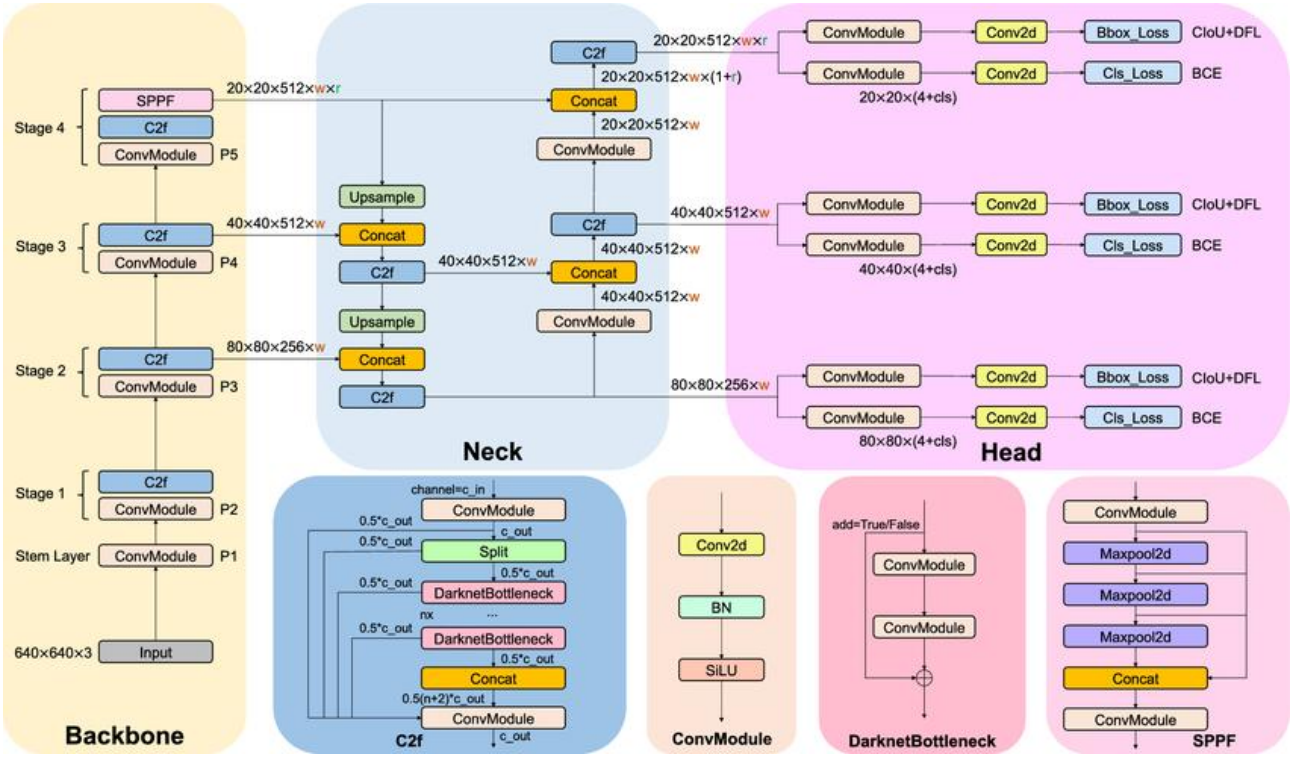

Рис. 3.14. - Узагальнена архітектура YOLOv8

3.3.2 Опис гіперпараметрів

YOLOv8s – це легковажна та потужна архітектура для задач детекції об’єктів, що надає безліч налаштувань для адаптації моделі під конкретні дані та завдання. Вірно підібрані гіперпараметри мають ключову роль у покращенні точності та стабільності моделі. Розглянемо основні гіперпараметри, які можна налаштувати для YOLOV8s, та їх вплив на навчання та продуктивність. Наявні гіперпараметри можно описати декількома типами: загальні параметри, гіперпараметри аугментації данних, гіперпараметри моделі, оптимізація, параметри виходів моделі. [61, 62]

До загальних параметрів можно віднести Learning Rate (темп навчання), Batch Size (розмір батчу), Epochs (епохи), Momentum (Момент імпульсу), Weight Decay (регулярізація ваг моделі).

Learning Rate визначає як швидко змінюються ваги моделі на кожному кроці оптимізації. Використання адаптивних оптимізаторів дозволяє автоматизовано корегувати темп навчання.

Batch Size визначає кількість зображень які обробляються одночасно і

обмежується лише потужністю обладнання на якому буде проходити навчання.

Epochs визначає кількість разів для проходження усього тренувального набору даних.

Momentum вказує вклад раніше пройдених кроків оптимізації в наступний крок.

Weight Decay регулярізує ваги моделі та штрафу занатно великі значення ваг моделі запобігаючи перенавчанню.

До гіперпараметрів аугментації даних можно віднести Mosaic Probability, MixUp Probability, Flip Probability, Scale та Color Augmentation. Ці параметри необхідні для збільшення різноманітності тренувального набору даних та покращення узагальнюючої здатності моделі.

Mosaic Probability це мозаїчна аугментація що поєднує декілька зображень у одне змінюючи їх розташування.

MixUp Probability це технологія змішування для зображень з урахуванням їх початкових рамок.

Flip Probability виконує відзеркалення, і допомагає моделі в опрацьовуванні симетричних об’єктів.

Scale виконує масштабування зобряження.

Color Augmentation змінює яркість, контраст або насиченість зображень для гнучкої варіативності даних.

До гіперпараметрів моделі відносять Input Image Size, Box loss Coefficients, Classification Loss Coefficients , IoU Threshold.

Input Image Shape вказує на розмір вхідних зображень тренувального набору даних які модель повинна опрацьовувати.

Box loss Coefficients це коефіцієнт похибки, що описує похибку координат рамок при навчанні моделі.

Classification Loss Coefficients це коефіцієнт похибки, що описує похибку класифікації об’єкта при навчанні моделі.

IoU Threshold це метрика що вимірює ступінь перетину між передбаченою рамкою та істинною анотацією. Та вказуеє на мінімальне значення IoU,

необхідне для того, щоб передбачена рамка вважалася коректним збігом із реальною анотацією.

До гіперпараметрів оптимізації відносять Scheduler, Gradient Clipping, Label smoothing та Dropout.

Scheduler є Планувальником змінює темп навчання по мірі навчання моделі.

Gradient Clipping це обмеження градієнтів, цей параметр допомагає запобігти вибуховому зростанню градієнтів.

Label Smoothing вказує на згладжування міток. Це зменшує впевненість моделі за допомогою додавання шумів до істинних міток.

Dropout це регулярізація що відповідає за відключення випадкових нейронів під час навчання.

До параметрів виходу моделі відносять Confidence Threshold та NMS (Non-Max Suppression) Threshold.

Confidence Threshold (Поріг впевненості). Мінімальна ймовірність, коли об'єкт вважається виявленим.

NMS (Non-Max Suppression) Threshold. Фільтрує прямокутники, що перекриваються, на основі IoU.

Налаштування гіперпараметрів відіграє вирішальну роль у успішному навчанні та застосуванні моделі YOLOv8s. Вибір оптимальних значень залежить від специфіки завдання, доступних обчислювальних ресурсів та розміру тренувального набору даних. Оптимальне налаштування гіперпараметрів дозволяє досягти високої якості детекції об'єктів за умови збереження продуктивності моделі.

Під час проектування моделі були обрані наступні значення для гіперпараметрів: epochs=10, imgsz=256, batch=8, augment=False. Усі інші параметри були залишені за замовчуванням згідно з базовим налаштуванням YOLOv8s. Оптимізатором обрано SGD з параметрами (lr=0.01, momentum=0.9) через те, шо він краще працює з велики об’ємами даних та більш раціонально навантажує пам’ять.

Оптимізатор SGD (Stochastic Gradient Descent) використовується для мінімізації функції втрат шляхом коригування вагів моделі на основі обчислених градієнтів.

SGD оновлює параметри моделі (ваги та зміщення) з використанням оцінки градієнта функції втрат на основі невеликих випадкових підмножин даних міні-батчів. Формула оновлення вагів зображенна на формулі 3.3.

$$W(t+1) = W(t) - \eta \cdot \nabla L((W)t) \tag{3.3}$$

Де $(W)t$ це поточні ваги моделі, η це темп навчання (Learning Rate) а $\nabla L((W)t)$ це градієнт функції втрат L відносно вагів wt.

Для удосконалення недоліків SGD також можно використовувати його використовуючи momentum за формулою 3.4.

$$V(t+1) = \mu V(t) - \eta \nabla L(W(t)), W(t+1) = W(t) + V(t+1) \tag{3.4}$$

Де μ це коэфіцієнт моменту.

Stochastic Gradient Descent є важливим інструментом у навчанні нейронних мереж завдяки своїй ефективності, гнучкості та простоті. Сучасні модифікації, такі як Momentum, значно покращують його продуктивність, що робить його конкурентоспроможним навіть серед більш складних оптимізаторів. [38, 39]

3.4 Навчання моделі

Для навчання моделі скористаємося PyCharm IDE. Створимо новий проект, вказавши місце його розташування та версію Python (рис 3.15). Для виконання завдання був використан Python версії 3.10.

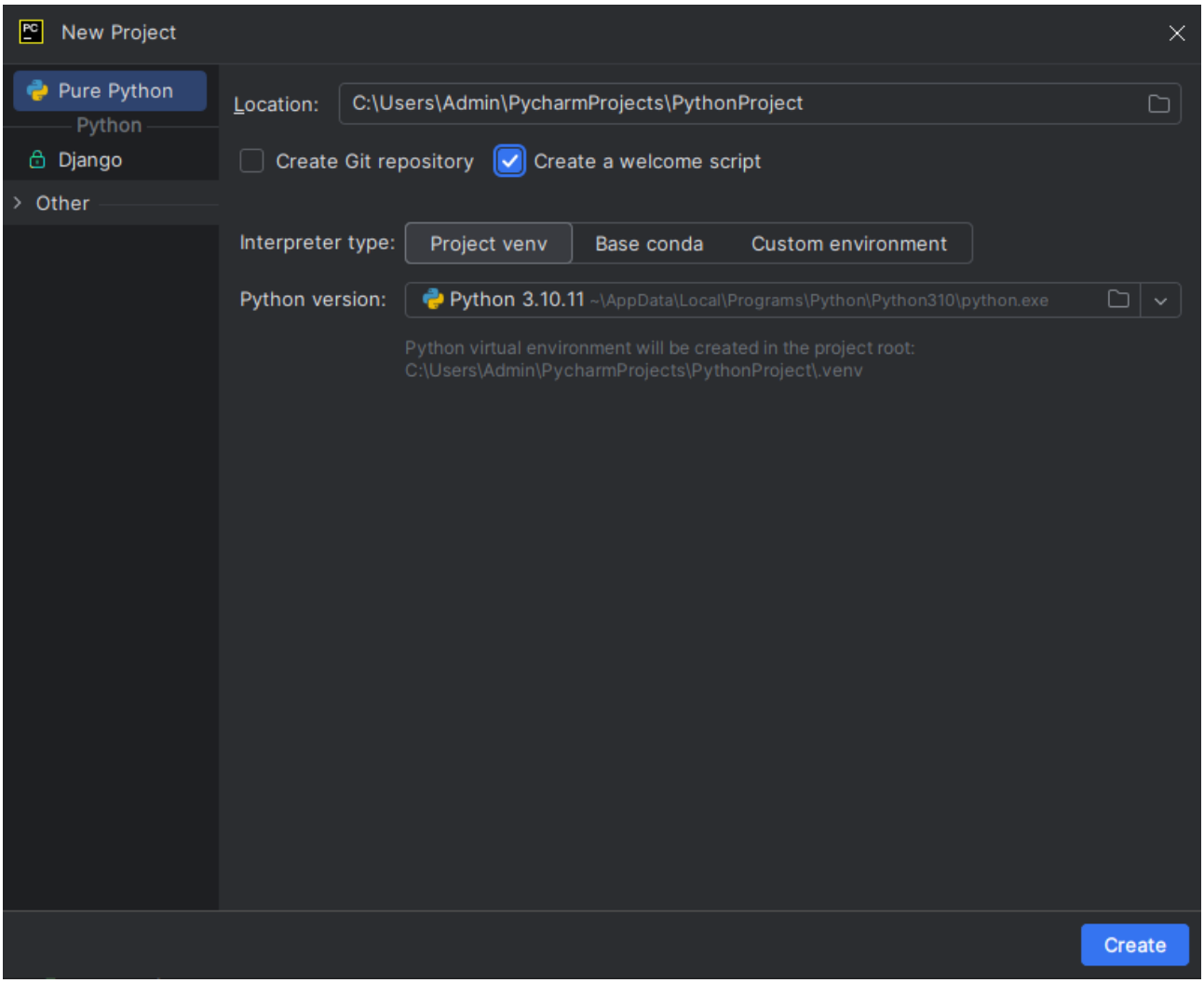

Рис. 3.15. - Вікно створення проекту PyCharm

Після того як файл проекту створено необхідно провести імпорти усіх потрібних бібліотек. Які перед цим необхідно інсталювати. Це можна зробити наступною командою що треба виконати у консолі.

pip install os python-yaml ultralytics torch

```
import os
import yaml
from ultralytics import YOLO
import torch
```

Після того як бібліотеки інстальовані треба прописати вибір девайсу на якому буде проводитись навчання. Це може бути CPU або GPU. У коді цей вибір автоматизовано за принципом доступності конкретного девайсу. Перевага надана GPU, проте у разі якщо це не є можливим автоматично використовується CPU.

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print(f"Using the device: {device}")
```

Після перевірки доступності девайсу також буде вказано який саме девайс буде використовуватися під час навчання. Також це значення фіксується у змінну і може зручно використовуватися надалі.

Далі треба вказати шляхи до основних директорій що будуть використовуватися. У коді це реалізовано наступним чином.

```
dataset_path = r'F:\New Plant Diseases Dataset(Augmented)'
output_dir = os.path.join(dataset_path, 'OUT')
if not os.path.exists(output_dir):
    os.makedirs(output_dir)
```

Де змінна dataset_path є основною і усі наступні файли та піддиректорії що будуть створюватися у ньому. Змінна output_dir вказує те що, віхідна директорія е піддиректоріє у dataset_path та має назву OUT. Далі у коді реалізована перевірка наявності вихідної директорії яка у разі її відсутності створить її самостійно.

Після цього треба вказати мапу класів. Нащо вона потрібна і як її створити було описано у другому розділі а саме у пункті 2.1.3, це можливо реалізувати як окремим файлом так і просто створенням словника у самому коді. Але що б все було «перед очима» і не було потреби переключень від файлу до файлу випадку потреби змінити щось це легше залишити у коді.

```
class_map = {
    'Apple___Apple_scab': 0,
    'Apple___Black_rot': 1,
    'Apple___Cedar_apple_rust': 2,
    'Apple___healthy': 3,
    'Blueberry___healthy': 4,
    'Cherry_(including_sour)___Powdery_mildew': 5,
    'Cherry_(including_sour)___healthy': 6,
    'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot': 7,
    'Corn_(maize)___Common_rust_': 8,
    'Corn_(maize)___Northern_Leaf_Blight': 9,
    'Corn_(maize)___healthy': 10,
    'Grape___Black_rot': 11,
    'Grape___Esca_(Black_Measles)': 12,
    'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)': 13,
    'Grape___healthy': 14,
    'Orange___Haunglongbing_(Citrus_greening)': 15,
    'Peach___Bacterial_spot': 16,
    'Peach___healthy': 17,
    'Pepper,_bell___Bacterial_spot': 18,
    'Pepper,_bell___healthy': 19,
    'Potato___Early_blight': 20,
    'Potato___Late_blight': 21,
    'Potato___healthy': 22,
```

```
    'Raspberry___healthy': 23,
    'Soybean___healthy': 24,
    'Squash___Powdery_mildew': 25,
    'Strawberry___Leaf_scorch': 26,
    'Strawberry___healthy': 27,
    'Tomato___Bacterial_spot': 28,
    'Tomato___Early_blight': 29,
    'Tomato___Late_blight': 30,
    'Tomato___Leaf_Mold': 31,
    'Tomato___Septoria_leaf_spot': 32,
    'Tomato___Spider_mites Two-spotted_spider_mite': 33,
    'Tomato___Target_Spot': 34,
    'Tomato___Tomato_Yellow_Leaf_Curl_Virus': 35,
    'Tomato___Tomato_mosaic_virus': 36,
    'Tomato___healthy': 37,
}
```

Наступним кororком треба зробити файл конфігурації data.yaml.

YAML (YAML Ain`t Markup Language) це файл який містить у собі усі відомості що необхідні для YOLO які він використовує при конфігурації. Цей файл має описувати структуру даних, кількість класів та їх назви.

```
data_yaml = {
    'train': os.path.join(dataset_path, 'train_yolo'),
    'val': os.path.join(dataset_path, 'val_yolo'),
    'nc': len(class_map),
    'names': list(class_map.keys())
}

yaml_path = os.path.join(output_dir, 'data.yaml')

with open(yaml_path, 'w') as yaml_file:
    yaml.dump(data_yaml, yaml_file)
```

Як і у випадку з мапою класів це реалізовано не окремо а одразу у коді. Необхідні директорії отримуються як шлях до відповідних директорій що були описані ще у самому початку, кількість класів атоматично зазначається як довжина карти классів, а назви як ключі до все ж с тієї ж мапи класів.

Далі новостворений файл конфігурації інпортується для подальшої роботи.

Так як мапа класів та файл конфігурації пов’язні і одне використовується при формуванні іншого рішення щодо їх розміщення не окремо а у коді дозволяє змінювати іх умовно напів-автоматично.

Після того як усі необхідні конфігураційні файли та шляхи до директорії були налаштовані необхідно імпортувати передбачену модель YOLOv8s.

```
model = YOLO("yolov8s.pt")
```

Вона автоматично інсталюється у директорію проекту. Також саме тут можливо змінити її на будь яку іншу з представлених офіційним розробником (YOLOv8n, YOLOv8s, YOLOv8m, YOLOv8l, YOLOv8x). Але також існує і інший варіант для імпорту передбаченої моделі.

```
MODEL_PATH = r" F:\New Plant Diseases
Dataset(Augmented)\OUT\yolov8_model\weights\last.pt"
model = YOLO(MODEL_PATH)
```

У цьому випадку показано варіант більш конкретного шляху до файлу передбаченої моделі. Таким шляхом можна використовувати чужі кастомні моделі або як прописано зараз використовувати файли, що сама ж YOLO створила під час навчання. Річ у тім, що за стандартних налаштувань у під час навчання YOLO зберігає два файли best.pt та last.pt і вони можуть бути корисними. Best.pt говорить сам за себе. Він має у собі найкращі ваги моделі за час навчання і потрібен під час фінального збереження моделі. А ось last.pt має ваги станом на останню епоху що була завершена і потрібна що б у разі якогось збою мати можливість продовжити навчання з цієї точки а не робити все з самого початку. Умовно якщо модель пройшла 7 епох з передбачених 10-и цей файл матиме ваги станом на 7-му епоху і при використанні такого імпорту та гіперпараметру resume модель почне з цієї точки запам'ятавши також те, що це саме 7-а епоха і завершить навчання так якби ніякого збою і не було.

Це особливо актуально через стабілізаційні вимкнення світла які наразі стають нормою в нашому житті. Навчання моделі займає великий час і тому у разі вимкнення світла це спасає увесь процес навчання без потреби у сильній переробці коду як це робилося б при використанні інших архітектур.

Тепер необхідно прописати тренування моделі вказавши потрібні гіперпараметри, якщо якісь з них не вказувати вони будить обрані згідно зі значеннями за замовченням.

```
model.train(
    data=yaml_path,
    epochs=10,
    imgsz=256,
    batch=8,
    project=output_dir,
    name='yolov8_model',
    save=True ,
    patience=5,
    verbose=True,
    augment=False,
    device=device,
    save_period=1,
    #resume=True,
)
```

Data передає шлях до конфігурації.

Epochs передає кількість епох. Скільки разів модель повинна пройти через увесь набір тренувальних даних.

Imgsz вказує розмір зображень які будуть використовуватися під час навчання. У разі якщо зображення не будут відповідати цьому значенню вони будут автоматично масштабуватись до необхідного розміру. Чим більшим буде це значення тим більш потужне обладнання необхідно.

Batch вказує на кількість зображень що будуть оброблятись під час навчання одночасно.

Project це шлях до директорії де будуть зберігатись логи, та інші данні.

Name це назва проекту, буде створено підкаталог у директорії проекту для зберігання результатів.

Save дозволяє або забороняє збереження моделі.

Patience відповідає за ранню зупинку навчання у разі якщо протягом вказаного числа епох не буде фіксуватись покращення результатів.

Verbose дозволяє або забороняє відображення прогресу навчання у консолі.

Augment дозволяє або забороняє виконання аугментації над тренувальними зображеннями датасету.

Device вказує на якому обладнані буде виконуватись навчання.

Save_period вказує частототу збережень проміжних результатів.

Resume дозволяє або забороняє продовження навчання з останної успішно

завершеної епохи.

Далі слід запустити код. YOLO почне формувати файл конфігурації, просканує усі файли, перевірить їх на коректність та розпочне процес навчання. Після початку навчання модель також сформує приклад тренувального батчу (рис. 3.16.).

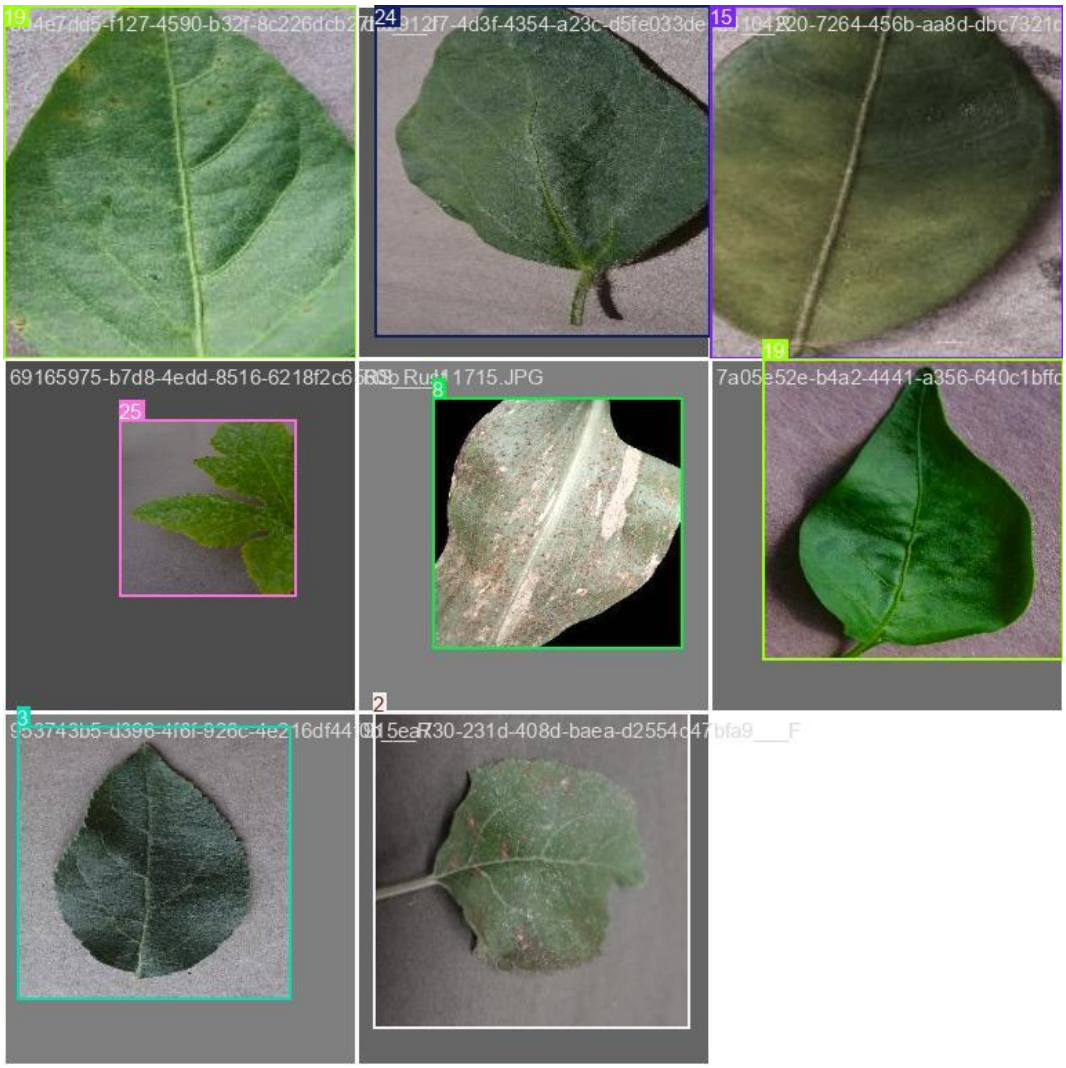

Рис. 3.16. - Приклад тренувального батчу.

Процес навчання зображено на рис. 3.17.

Рис. 3.17. - Приклад процесу навчання.

Після завершення процесу навчання YOLO самостійно сформує звіти та складе усі необхідні графіки для оцінки якості моделі.

3.5 Розробка веб застосунку для роботи з моделлю

Розроблений веб застосунок призначений для виявлення об'єктів на завантажуваних зображеннях з використанням кастомної моделі YOLOv8. Ціль проекту це створення простої тестової програми для обробки зображень з можливістю візуалізації результатів та отримання статистики по виявленим об’єктам.

Цей застосунок не є фінальною ціллю а створюється виключно для тестової демонстрації роботи моделі. У подальшому використанні моделі треба буде інтегрувати її в вже існуючі системи, або допрацюванні цього застосунку.

Додаток дозволяє користувачу завантажувати зображення, обробляти його за допомогою навчаної моделі YOLOv8 і отримувати наступні результати: відображення опрацьованого зображення з візуалізацією найдених об’єктів, статистику детекції щодо того які саме об’єкти були знайдені та їх кількість.

Застосунок складається з таких ключових компонентів: підготовка моделі, завантаження та попередня обробка зображення, опрацювання зображення моделлю та генерація статистики, інтерфейс та відображення результату, можливість зберігання обробленого зображення.

До підготовки моделі входить її імпорт.

```
MODEL_PATH = r"шлях до файлу моделі"
model = YOLO(MODEL_PATH)
```

Попередня обробка зображення включає у себе перетворення отриманого зображення у квадратне з додаванням полів. Поля потрібні для збереження пропорцій, а саме по собі перетворення не є обов'язковим так як моделі YOLO вміють працювати з різними зображеннями.

```
def pad_to_square(image):
    width, height = image.size
    max_side = max(width, height)
    delta_w = max_side - width
    delta_h = max_side - height
    padding = (delta_w // 2, delta_h // 2, delta_w - delta_w // 2, delta_h -
delta_h // 2)
    new_image = ImageOps.expand(image, padding, fill=(0, 0, 0))  # Black padding
    return new_image
```

Далі перетворене зображення передається до моделі де воно опрацьовується та формується статистика.

```
@app.route("/", methods=["GET", "POST"])
def index():
    global processed_image, detection_stats
    image_url = None

    if request.method == "POST":
        file = request.files["image"]
        input_image = Image.open(file.stream)
        squared_image = pad_to_square(input_image)
        results = model(squared_image)
        processed_image_array = results[0].plot()
        processed_image_array = cv2.cvtColor(processed_image_array,
cv2.COLOR_BGR2RGB)
        processed_image = Image.fromarray(processed_image_array)
        detection_stats = {}
        for box in results[0].boxes:
            cls_name = results[0].names[int(box.cls)]
            detection_stats[cls_name] = detection_stats.get(cls_name, 0) + 1
        buffer = io.BytesIO()
        processed_image.save(buffer, format="JPEG")
        buffer.seek(0)
        image_base64 = base64.b64encode(buffer.getvalue()).decode("utf-8")
        image_url = f"{image_base64}"

    return render_template_string(HTML_TEMPLATE, image_url=image_url,
stats=detection_stats)
```

Можливість завантаження опрацьованого зображення реалізовано наступним чином:

```
@app.route("/download")
def download():
    global processed_image

    if processed_image is not None:
        # Send the processed image as a file for download
        buffer = io.BytesIO()
        processed_image.save(buffer, format="JPEG")
        buffer.seek(0)
        return send_file(buffer, as_attachment=True,
```

```
download_name="processed_image.jpg", mimetype="image/jpeg")
    else:
        return redirect(url_for("index"))
```

Крім цього HTML шаблон, що необхідний для роботи цього застосунку теж знаходиться у коді а не у окремому файлі та має наступний вигляд:

```
HTML_TEMPLATE = """
<!doctype html>
<html lang="en">
<head>
    <title>YOLO Detection</title>
</head>
<body>
    <h1>Upload an image</h1>
    <form method="post" enctype="multipart/form-data">
        <input type="file" name="image" accept="image/*" required>
        <button type="submit">Process</button>
    </form>
    {% if image_url %}
        <h2>Result</h2>
        <img src="{{ image_url }}" alt="Processed image">
        <h3>Statistics:</h3>
        <ul>
        {% for cls, count in stats.items() %}
            <li>{{ cls }}: {{ count }}</li>
        {% endfor %}
        </ul>
        <br>
        <a href="{{ url_for('download') }}">Download processed image</a>
        <br><br>
        <form method="get" action="{{ url_for('index') }}">
            <button type="submit">Back</button>
        </form>
    {% endif %}
</body>
</html>
"""
```

Інтерфейс застосунку та приклад його роботи зображено на рис. 3.18, 3.19.

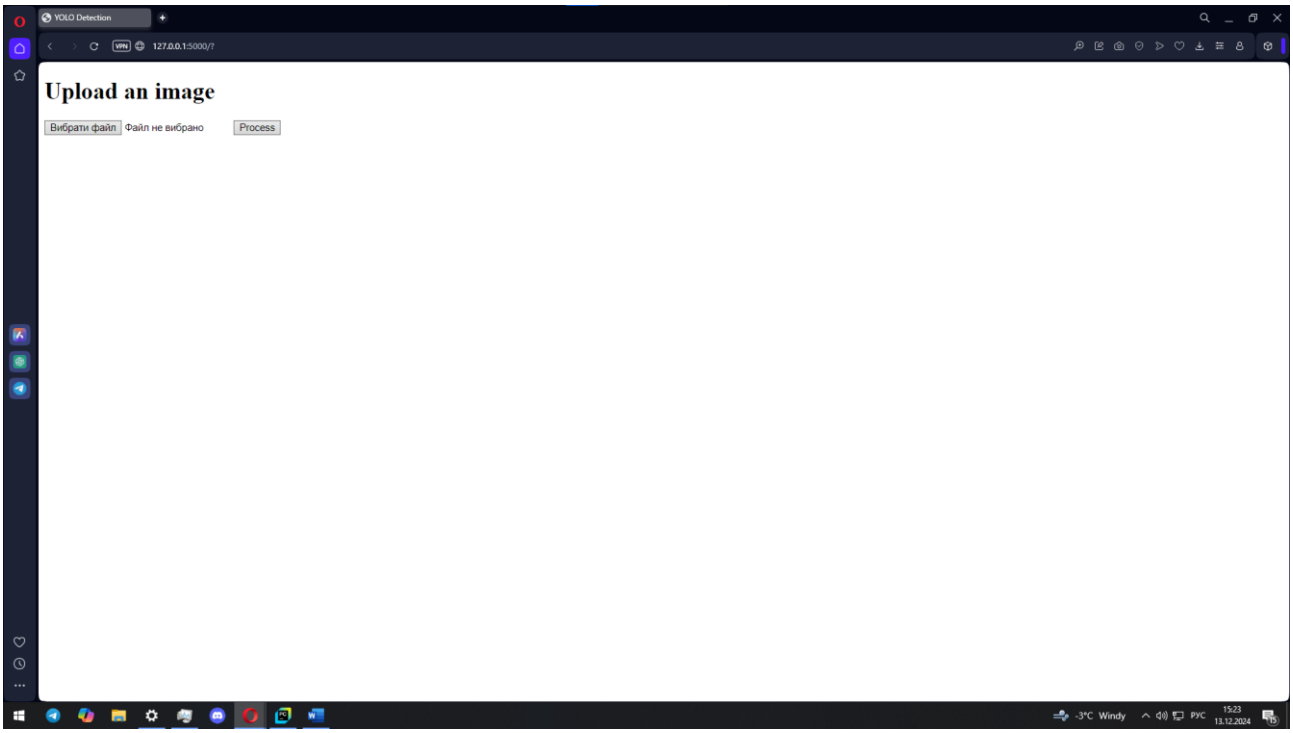

Рис. 3.18. - Інтерфейс застосунку.

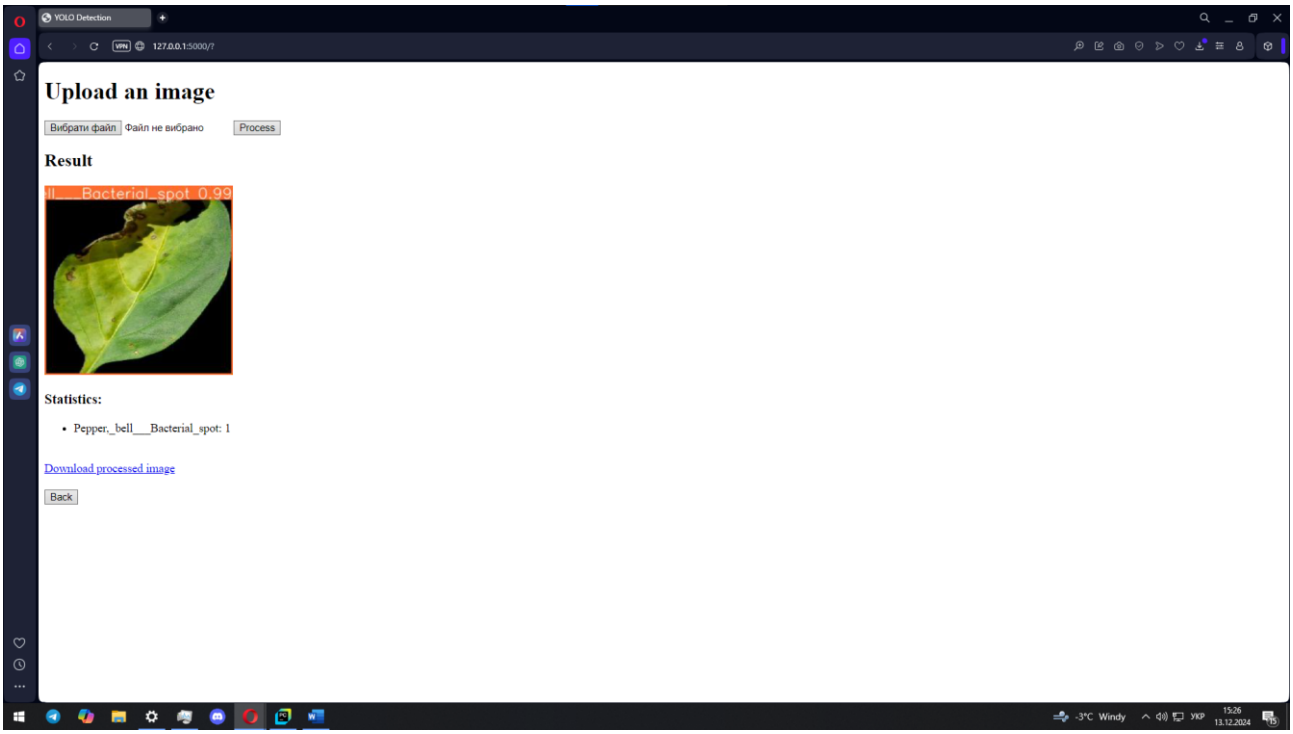

Рис. 3.19. - Приклад роботи застосунку.

Можливості для доработки - розширення функціональності, підтримка обробки відеофайлів, додавання моделі конфігурації через інтерфейс користувача, але цей додаток вже зараз дозволяє у тестовому варіанті скористатися моделлю та на власні очі побачити те як вона працює.

3.6 Висновок

У розділі оглянуто та проаналізовано програмні рішення шо допоможуть при проектуванні моделі.

Як основна архітектура для розробки проекту обрана YOLOv8. Хоча базово на момент виходу вона і програвала Faster R-CNN у точності, але восьма версія що була представлена у 2023-у році значно покращилась з моменту першої версії YOLO і може бути конкурентною навіть з Faster R-CNN. Також ця архітектура є значтно простішою у використанні та біль швидшою, що дозволить використовувати розробляєму модель як у реальному часі, так і на автономних станціях з вирощування рослин.

Python є лідером у направленні глибокого машинного навчання та був обраний за основу через свою простоту та широку інформаційну базу. Також розглянуто основні бібліотеки та фреймворки що можуть використовуватися.

LebelImg також є одним з най популярніших виборів для задачі ручного анотування та розмітки зображень, що допоможе у формуванні датасету.

Докладно описано вибір та необхідні дії для створення датасету, обгрунтування вибору моделі, а також налаштування гіперпараметрів для вирішення поставленого завдання.

Підбір гіперпараметрів здійснювався з урахуванням особливостей завдання та структури даних, що дозволило досягти оптимального балансу між точністю та продуктивністю моделі. В результаті модель адаптована для ефективного вирішення поставленого завдання.

РОЗДІЛ 4 АПРОБАЦІЯ ТА ТЕСТУВАННЯ

4.1 Огляд результатів навчання

Під час навчання моделей YOLOv8 звітні дані надають інформацію про ефективність моделі та якість навчання. Ця звітність складається з таких діаграм і графіків як: матриця Помилок, F1-Крива, Precision-Крива, Recall-Крива та PR-Крива.

4.1.1 Опис матриці помилок

Матриця помилок - це таблиця, що показує, як класифікатор розподіляє передбачення проти істинних класів даних. У багатокласовій класифікації матриця помилок має розмір NxN де N – кількість класів. Кожен рядок матриці представляє дійсні класи, а кожен стовпець — передбачені класи. Приклад такої структури зображено на таблиці 4.1. [65]

Класи	Клас A	Клас B	Клас C	…
Клас A	TP(A)	FP(A->B)	FP(A->C)	…
Клас B	FN(F->B)	TP(B)	FP(B->C)	…
Клас C	FN(A->C)	FN(B->C)	TP(C)	…
…	…	…	…	…

Таблиця 4.1. - Приклад матриці помилок.

Діагональні елементи таблиці (TP) це ті об'єкти які моделі правильно класифікувала. Не діагональні елементи таблиці (FP, FN) це помилки класифікації. Наприклад FN(A->B) вказує кількість об'єктів класу A які помилково були зазначені як клас B.

Матриця що була створена на основі навчання розробляємої моделі зображена на рис. 4.1, нормалізована версія зображена на рис. 4.2.

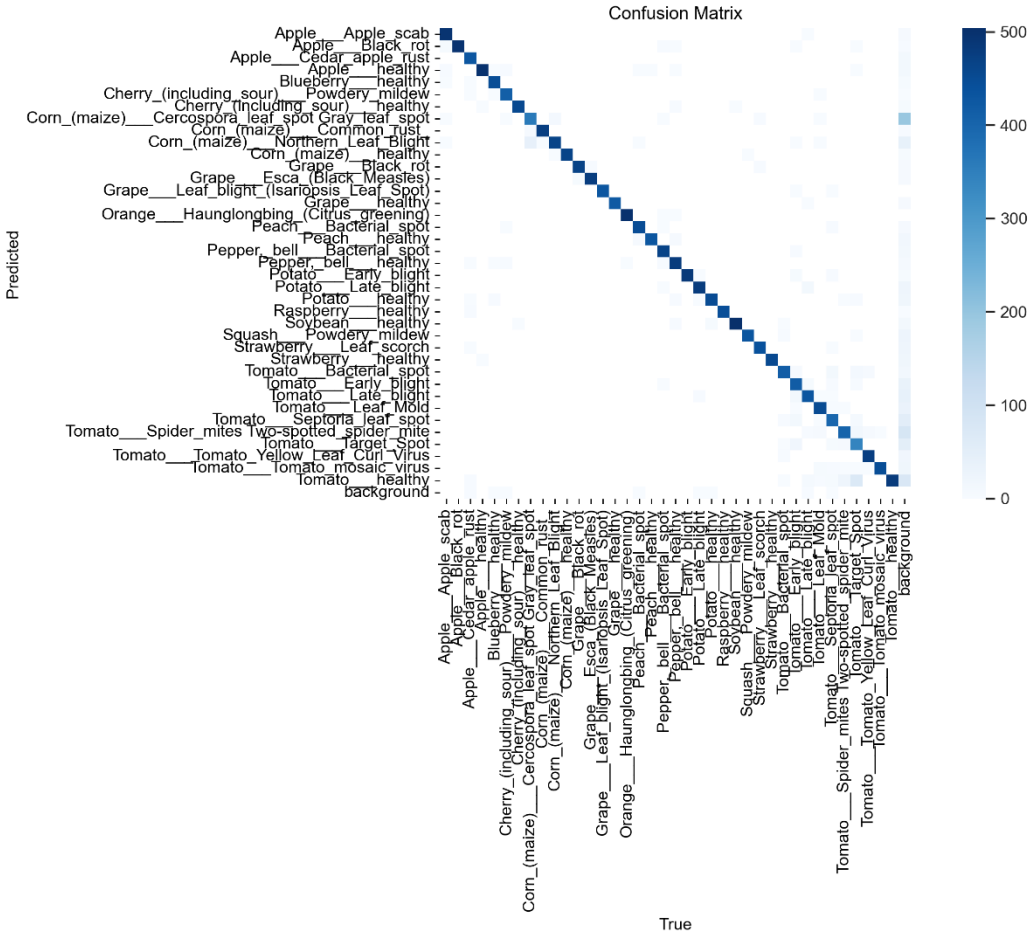

Рис. 4.1. - Матриця помилок.

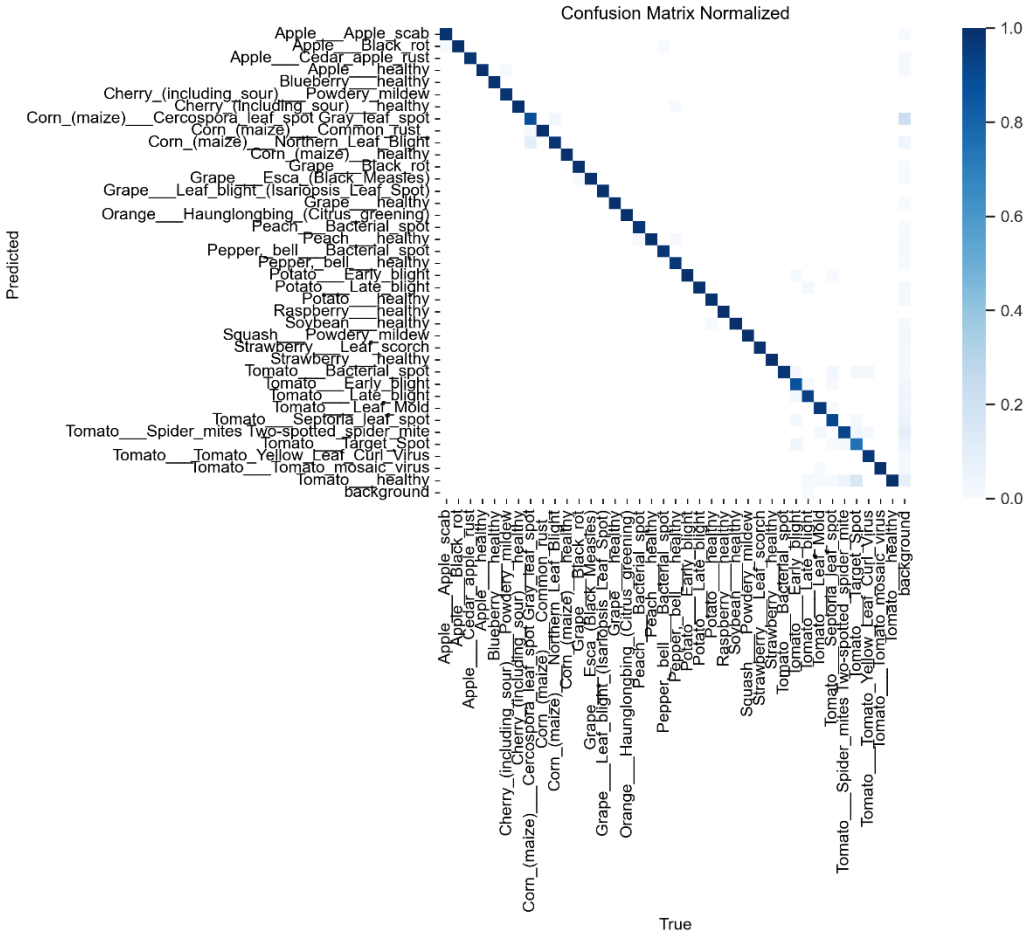

Рис. 4.2. - Нормалізована матриця помилок.

Цей графік вказує нам на те, наскільки коректно поводиться модель. Загалом показує хороший результат, однак, має деякі незначні труднощі з визначенням фону.

4.1.2 Опис F1-кривої

F1-Крива. це графік, що відображає значення F1-міри при зміні порога класифікації. Ось X - поріг ймовірності від 0 до 1, ось Y це значення F1-міри. F1-міра поєднує Precision та Recall (Точність та повнота) та вираховується за формулою 4.1. [63, 64]

$$\mathrm{F1} = 2 \cdot \frac{Precision * Recall}{Precision + Recall} \tag{4.1}$$

F1-крива показує, при якому порозі класифікації досягається найкращий баланс між Precision та Recall.

F1-крива для навченої моделі зображена на рис. 4.3.

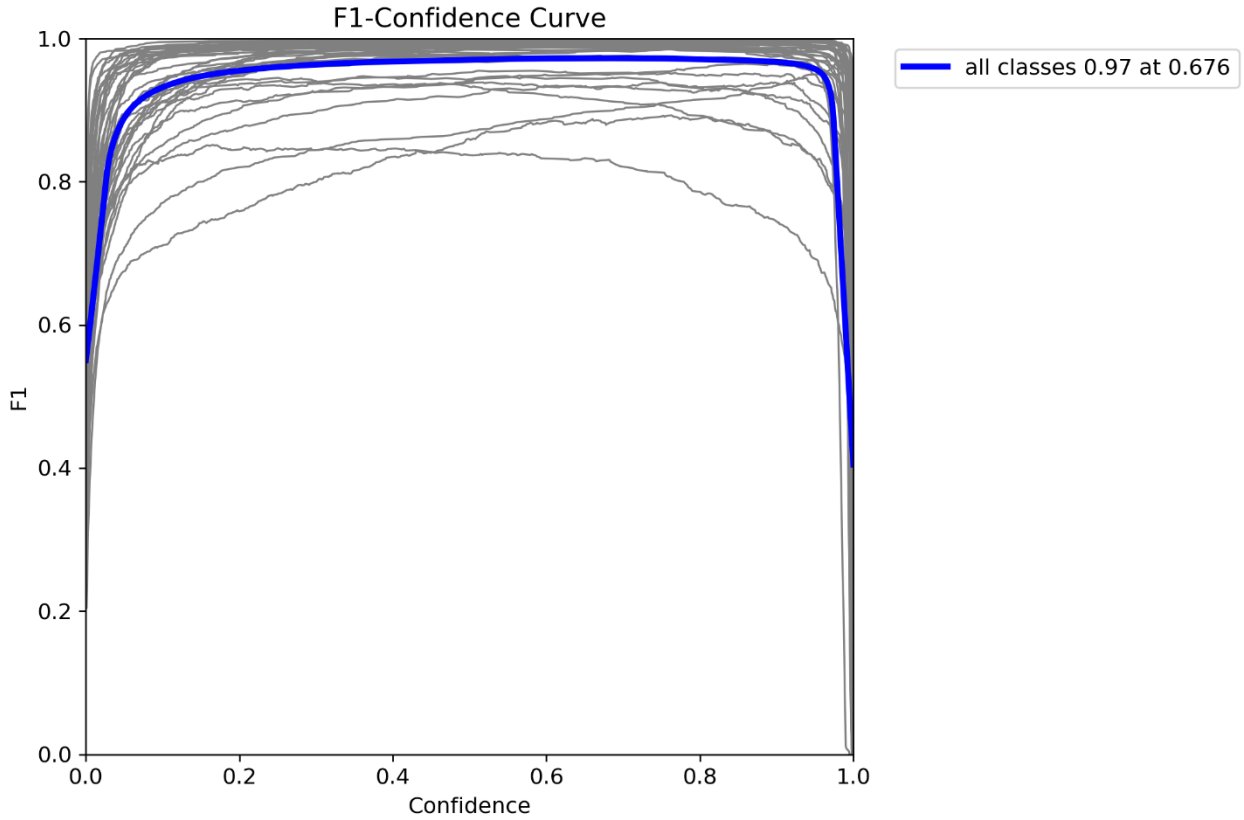

Рис. 4.3. - F1-крива.

Сірим кольором показано відповідна крива для кожного з класів. Сине - крива для усіх класів. Найкращий показник F1-міри при цьому досягається при значенні Confidence (Впевненість) = 0.676.

4.1.3 Опис Precision-кривої

Precision-крива (точність) є показником того як саме змінюється точність в залежності від погору класифікації. Ось X - поріг ймовірності, ось Y - значення точності. Точність вираховується за формулою 4.2 і зображено на рис. 4.4. [63]

$$Precision = \frac{TP}{TP + FP} \quad (4.2)$$

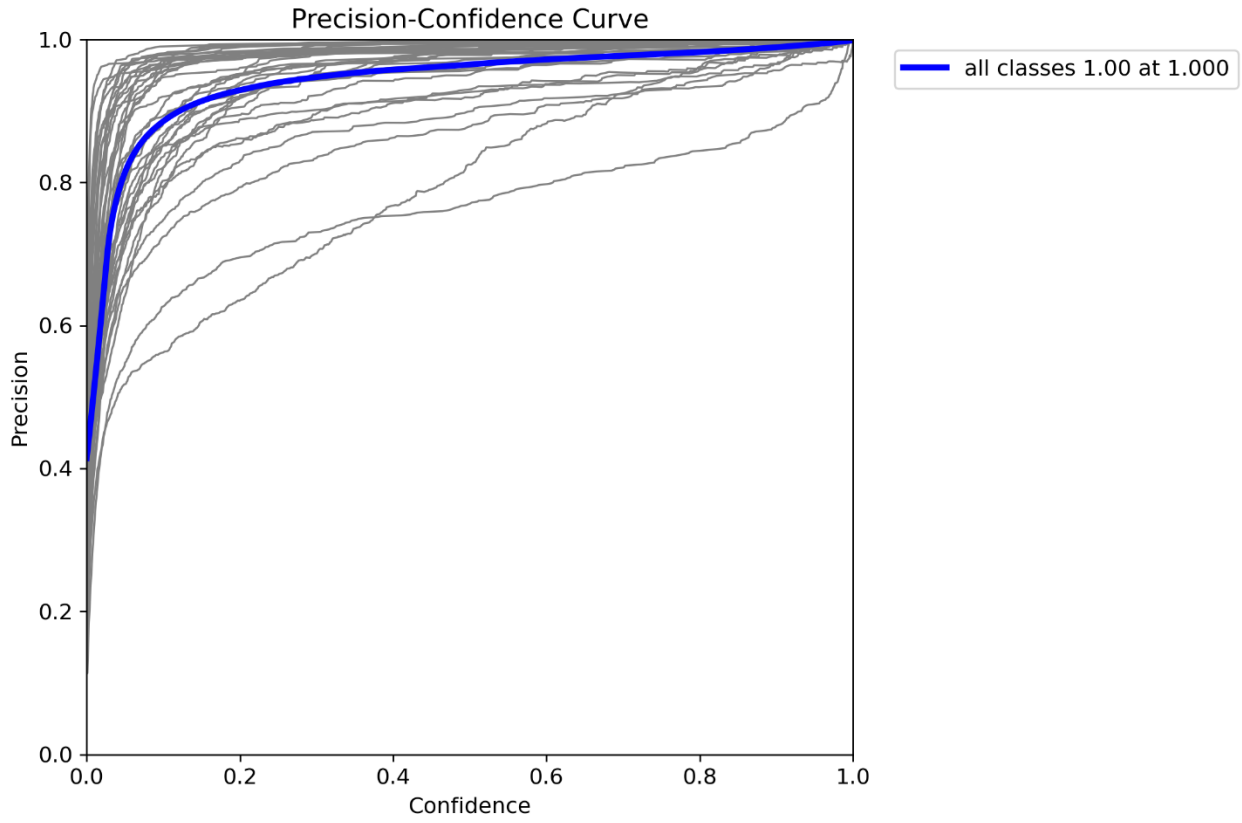

Рис. 4.4. - Precision-крива.

Сірим кольором показано відповідна крива для кожного з класів. Сине - крива для усіх класів. Хоча згідно графіку найкраще значення досягається при значенні чутливості 1.0 це не є показником того, що це оптимальне значення для чутливості.

4.1.4 Опис Recall-кривої

Recall-крива (повнота) показує, як змінюється повнота передбачень (Recall) в залежності від порогу класифікації. Ось X - поріг ймовірності, ось Y - значення Recall. Recall показує долю справді позитивних об'єктів, які модель вдалося знайти. Recall вираховують за формулою 4.3, а Recall-крива зображена на рис. 4.5. [63, 64]

$$Recall = \frac{TP}{TP + FN} \tag{4.3}$$

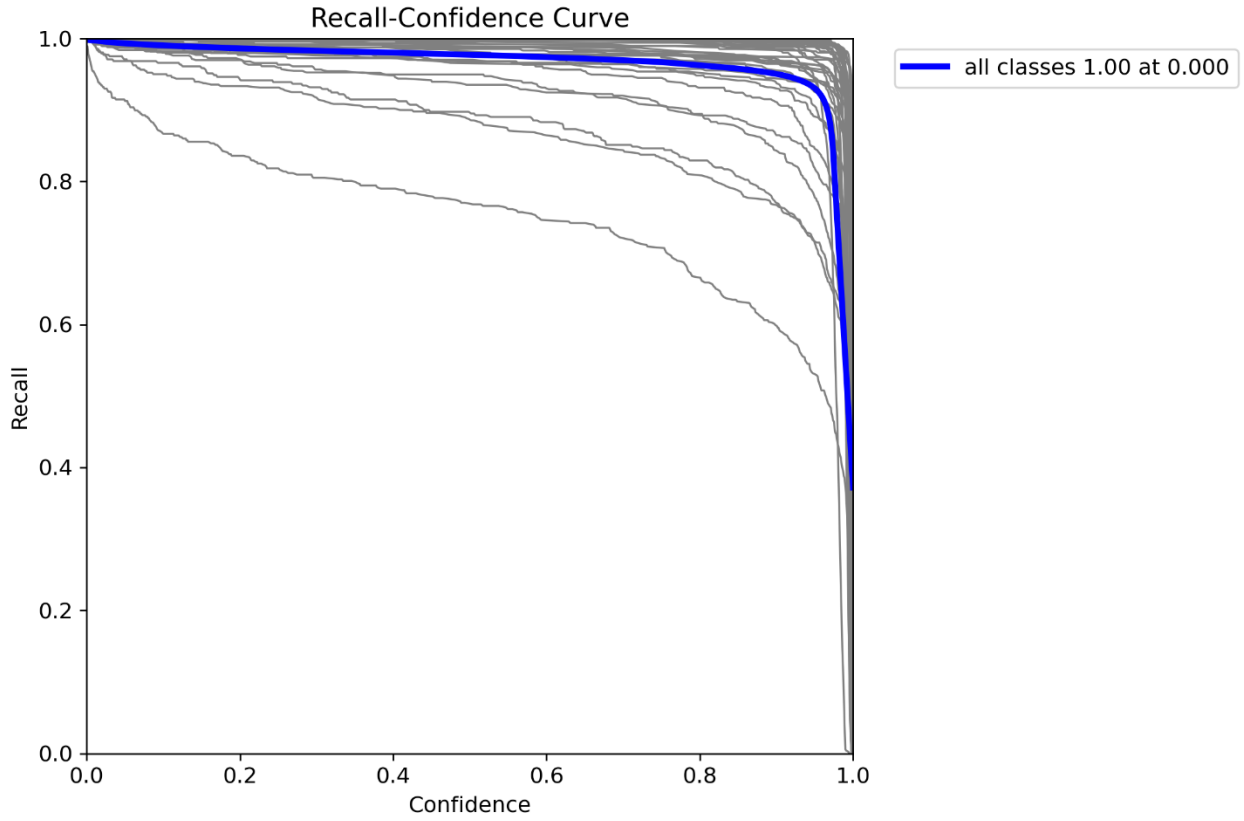

Рис. 4.5. - Recall-крива.

Сірим кольором показано відповідна крива для кожного з класів. Сине - крива для усіх класів. Чим нижчий поріг ймовірності тим вище значення Recall. Це пов’язано з тим, що модель починає класифікувати все більше об’єктів як вірні з урахуванням раніше пропущених. Як і для Precision найкраще значення при чутливості 0.0 не вказує на те, що такая чутливість насправді оптимальна.

4.1.5 Опис PR-кривої

PR-крива це Precision-Recall крива. Ця крива візуалізує залежність точності під повноти при змінах порогу класифікації. Ось X - повнота, ось Y - точність. PR-крива зазвичай використовується для оцінки моделей з незбалансованим набором даних і в відмінності від ROC кривої показує зміни для позитивного класу, що важливо, якщо домінує негативний клас. Ця крива показана на рис. 4.6. [63, 64, 66]

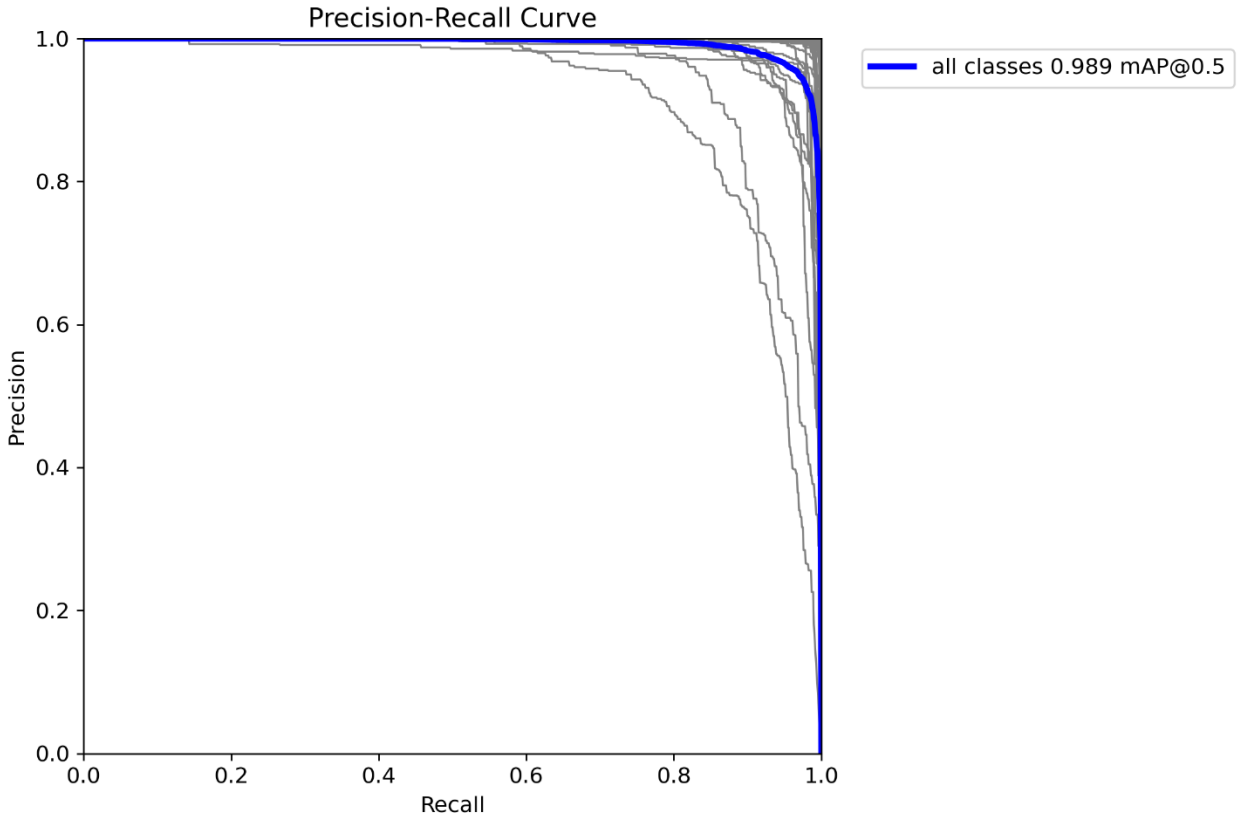

Рис. 4.6. - PR-крива.

Сірим кольором показано відповідна крива для кожного з класів. Сине - крива для усіх класів. Найкраща модель прагне до верхнього правого вугла (висока точність і повнота). mAP (mean Average Precision) - середня точність за всіма класами. Це основна метрика для оцінки якості моделей у задачах виявлення об'єктів. 0.5 вказує, що модель оцінюється при фіксованому поріг IoU = 0,5. IoU вимірює ступінь перекриття предсказанной рамки з істинною рамкою. Значення 0.5 означає, що передсказана рамка вважається правильною, якщо вона перекривається з істинною рамкою мінімум на 50%.

Враховуючи цей графік можна зробити висновок, що розроблена модель має показник точності 98.9% при значенні IoU 0,5, що відповідає сучасним вимогам.

4.1.6 Валідаційні батчі сформовані моделлю

Крім цього YOLO також зберігає приклад валідаційних зображень. Приклад такого зображення можна побачити на рис. 4.7, 4.8.

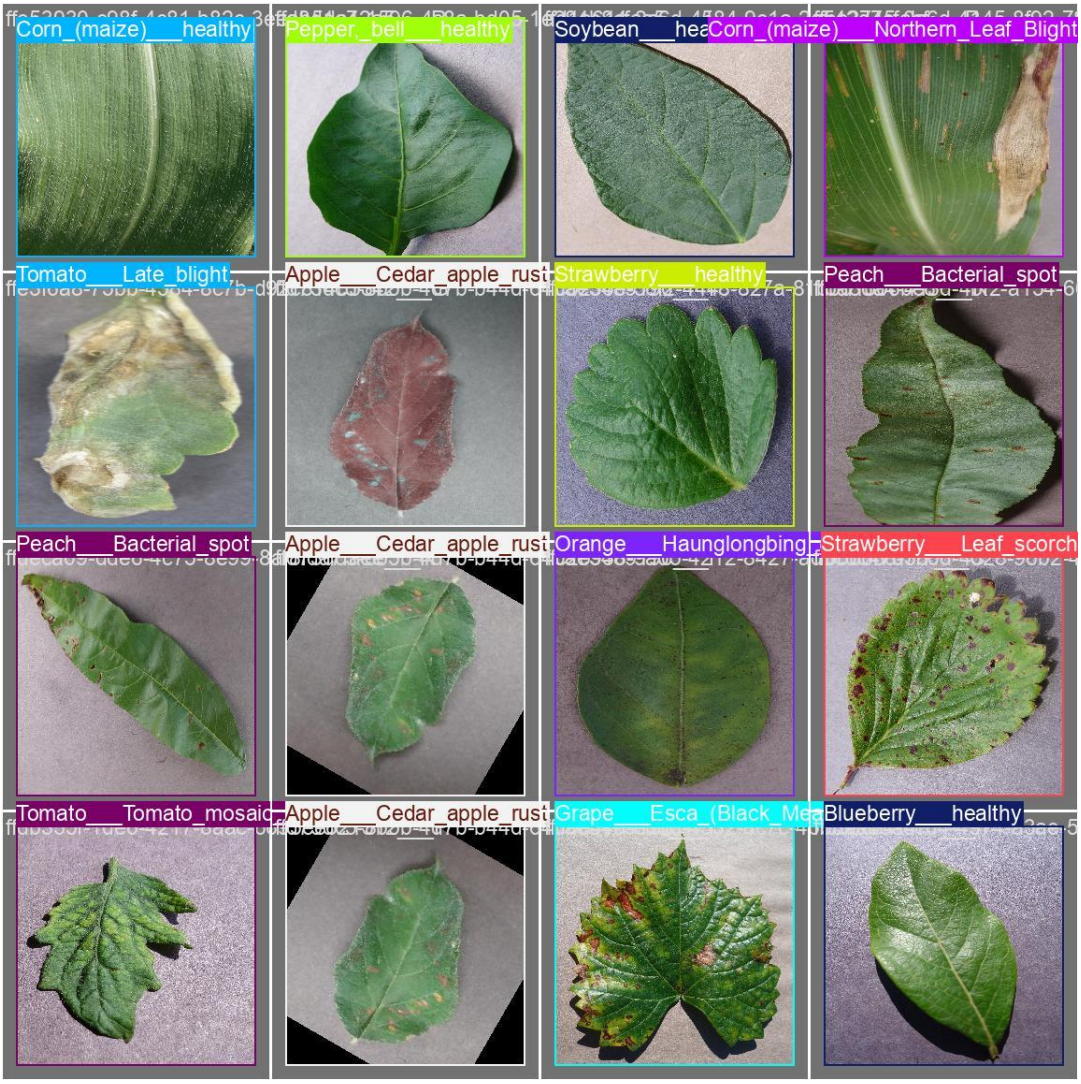

Рис. 4.7. - Істинний набір зображень.

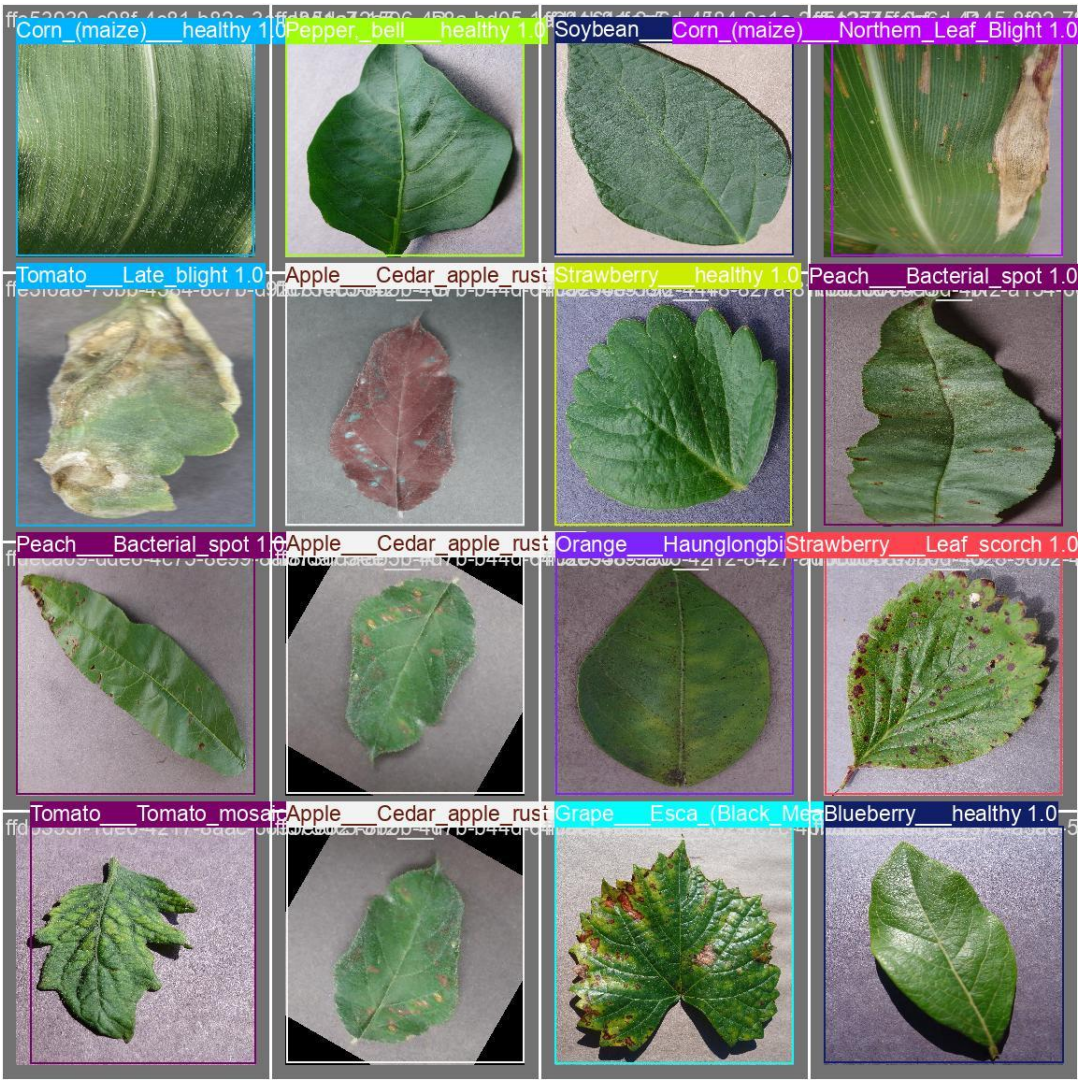

Рис. 4.8. - Результат роботи моделі.

ВИСНОВКИ

У даній роботі проведено грунтовне дослідження сучасних методів та технологій для автоматизації діагностики стану рослин на основі аналізу зображень. Виконані дослідження та розробки дозволяють зробити такі висновки:

Проведено детальний огляд архітектур нейронних мереж, що застосовуються для роботи з візуальними даними. Зокрема, розглянуто переваги та обмеження згорткових нейронних мереж (CNN), таких як LeNet, AlexNet, VGGNet та ResNet. Також проаналізовано об'єктні детектори (YOLO, Faster R-CNN, SSD), сегментаційні нейронні мережі (U-Net, Mask R-CNN), трансформери (Vision Transformer, Swin Transformer) і гібридні моделі. Це дозволило визначити найбільш ефективний підхід для поставленої задачі.

Для вирішення задачі автоматичної діагностики рослин обрана архітектура YOLOv8 завдяки її високій продуктивності, здатності працювати в реальному часі, простоті інтеграції та оптимальному співвідношенню точності і швидкості. Порівняно з іншими методами, YOLOv8 демонструє високу ефективність у завданнях детекції, а її остання версія включає значні поліпшення у швидкості обчислень та зручності використання.

Виконано формування високоякісного датасету, що включає зображення здорових рослин та рослин із ознаками захворювань. Для цього застосовано інструмент LabelImg, який забезпечив анотацію зображень у необхідному форматі для використання в YOLOv8. Додатково реалізовано автоматизацію створення карти імен класів, що спростило інтеграцію даних у модель.

Здійснено налаштування гіперпараметрів моделі, таких як швидкість навчання, розмір батчу, кількість епох, а також параметри аугментації даних. Це дозволило підвищити загальну якість та стабільність навчання, забезпечивши адаптацію моделі до специфіки даних. Крім того, для навчання застосовано сучасні інструменти Python, включаючи бібліотеки PyTorch, Ultralytics та TensorFlow, що сприяло ефективній реалізації розробки.

Проведено тестування моделі на валідаційному наборі даних, що

дозволило оцінити точність класифікації, детекції та сегментації рослин і хвороб. Модель показала високі результати, що підтверджують її придатність до використання у реальних умовах. Відповідно до отриманих метрик, модель може бути інтегрована у сільськогосподарські системи для покращення ефективності моніторингу стану рослин.

Розроблене рішення може бути застосоване у широкому спектрі завдань, включаючи використання як підтримку автономних систем управління сільським господарством, а також у системах для садівників-аматорів. Його впровадження сприятиме зниженню витрат на догляд за рослинами, підвищенню врожайності та зменшенню втрат через несвоєчасну діагностику захворювань.

Таким чином, робота відкриває перспективи для впровадження інноваційних технологій у традиційні сільськогосподарські процеси, що дозволить створити більш стійку та ефективну агросферу в умовах глобального зростання населення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

2. Russakovsky, O., et al. (2015). "ImageNet Large Scale Visual Recognition Challenge." *International Journal of Computer Vision*.

3. Gonzalez, R., & Woods, R. (2018). *Digital Image Processing*. Pearson.

4. Van Der Walt, S., et al. (2014). "Scikit-image: Image processing in Python." *PeerJ*.

5. Wu, J., et al. (2020). "Comprehensive review on deep learning-based methods for image processing in agriculture." *Computers and Electronics in Agriculture*.

6. Sharma, A., et al. (2019). "Applications of image analysis in precision agriculture." *Agricultural Systems*.

7. Ma, L., et al. (2019). "Deep learning in remote sensing applications: A meta-analysis." *ISPRS Journal of Photogrammetry and Remote Sensing*.

8. Nex, F., & Remondino, F. (2014). "UAV for 3D mapping applications: A review." *Applied Geomatics*.

9. Maimaitijiang, M., et al. (2020). "Crop monitoring using satellite and UAV data." *Remote Sensing of Environment*.

10. Zeng, Y., et al. (2020). "Applications of UAV in precision agriculture." *Agronomy*.

11. Li, Y., et al. (2021). "Advances in satellite-based crop monitoring." *Earth Science Reviews*.

12. Toth, C., & Jóźków, G. (2016). "Remote sensing platforms and sensors: A survey." *ISPRS Journal of Photogrammetry and Remote Sensing*.

13. Krizhevsky, A., Sutskever, I., & Hinton, G. (2012). "ImageNet classification with deep convolutional neural networks." *Advances in Neural Information Processing Systems (NeurIPS)*.

14. Redmon, J., et al. (2016). "You Only Look Once: Unified, Real-Time Object Detection." *CVPR*.

15. Ren, S., et al. (2015). "Faster R-CNN: Towards Real-Time Object Detection."

Neural Information Processing Systems.

16. Szegedy, C., et al. (2015). "Going deeper with convolutions." *CVPR*.

17. Girshick, R., et al. (2014). "Rich feature hierarchies for accurate object detection and semantic segmentation." *CVPR*.

18. Tan, M., & Le, Q. (2019). "EfficientNet: Rethinking model scaling for convolutional neural networks." *ICML*.

19. Everingham, M., et al. (2010). "The Pascal Visual Object Classes (VOC) challenge." *International Journal of Computer Vision*.

20. Lin, T.-Y., et al. (2014). "Microsoft COCO: Common Objects in Context." *ECCV*.

21. Zhu, P., et al. (2019). "Object detection datasets: A survey." *Pattern Recognition Letters*.

22. Deng, J., et al. (2009). "ImageNet: A large-scale hierarchical image database." *CVPR*.

23. Yao, Z., et al. (2020). "Open Images: A public dataset for large-scale multi-label and multi-class image classification." *NeurIPS*.

24. Mohanty, S. P., et al. (2016). "Using deep learning for image-based plant disease detection." *Frontiers in Plant Science*.

25. Ferentinos, K. P. (2018). "Deep learning models for plant disease detection and diagnosis." *Computers and Electronics in Agriculture*.

26. Truong, N., et al. (2020). "AI-based systems for precision agriculture." *Sensors*.

27. Kouadio, L., et al. (2021). "Machine learning for greenhouse management systems." *Journal of Agricultural Engineering*.

28. Zhang, Q., et al. (2019). "Smart agriculture technologies." *IEEE Access*.

29. Su, J., et al. (2021). "Satellite technologies for precision agriculture." *Remote Sensing*.

30. Drusch, M., et al. (2012). "Sentinel-2: ESA's optical high-resolution mission." *Remote Sensing of Environment*.

31. Verrelst, J., et al. (2015). "Optical remote sensing for vegetation monitoring:

Sentinel-2 perspectives." *Remote Sensing*.

32. Roy, D. P., et al. (2014). "Landsat-8: Science and product vision." *Remote Sensing of Environment*.

33. Wulder, M. A., et al. (2019). "Landsat continuous monitoring." *Science of Remote Sensing*.

34. Hung, C., et al. (2019). "Automated monitoring systems for hydroponics." *Computers and Electronics in Agriculture*.

35. Gilmour, C., et al. (2020). "Hydroponic systems and automation in agriculture." *Journal of Horticultural Science*.

36. Solomon, C., & Breckon, T. (2011). *Fundamentals of Digital Image Processing*. Wiley.

37. Pratt, W. K. (2007). *Digital Image Processing: PIKS Inside*. Wiley.

38. Dalal, N., & Triggs, B. (2005). "Histograms of oriented gradients for object detection." *CVPR*.

39. Felzenszwalb, P. F., et al. (2010). "Object detection with discriminatively trained part-based models." *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

40. Hartley, R., & Zisserman, A. (2004). *Multiple View Geometry in Computer Vision*. Cambridge University Press.

41. Faugeras, O., & Luong, Q.-T. (2001). *The Geometry of Multiple Images*. MIT Press.

42. Zeiler, M. D., & Fergus, R. (2014). "Visualizing and understanding convolutional networks." *ECCV*.

43. LeCun, Y., et al. (1998). "Gradient-based learning applied to document recognition." *Proceedings of the IEEE*.

44. Simonyan, K., & Zisserman, A. (2015). "Very Deep Convolutional Networks for Large-Scale Image Recognition." *ICLR*.

45. He, K., et al. (2016). "Deep residual learning for image recognition." *CVPR*.

46. Liu, W., et al. (2016). "SSD: Single Shot MultiBox Detector." *ECCV*.

47. Ronneberger, O., et al. (2015). "U-Net: Convolutional Networks for Biomedical Image Segmentation." *MICCAI*.

48. He, K., et al. (2017). "Mask R-CNN." *ICCV*.

49. Dosovitskiy, A., et al. (2021). "An Image is Worth 16x16 Words: Transformers for Image Recognition." *ICLR*.

50. Liu, Z., et al. (2021). "Swin Transformer: Hierarchical Vision Transformer." *ICCV*.

51. Van Rossum, G., & Drake, F. L. (2009). *Python 3 Reference Manual*.

52. Abadi, M., et al. (2016). "TensorFlow: A system for large-scale machine learning." *OSDI*.

53. Paszke, A., et al. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library." *NeurIPS*.

54. Chollet, F. (2015). "Keras: Deep Learning for Humans." (GitHub).

55. Pedregosa, F., et al. (2011). "Scikit-learn: Machine Learning in Python." *Journal of Machine Learning Research*.

56. Hunter, J. D. (2007). "Matplotlib: A 2D Graphics Environment." *Computing in Science & Engineering*.

57. Waskom, M. L. (2021). "Seaborn: Statistical data visualization." *Journal of Open Source Software*.

58. JetBrains. (2022). *PyCharm Documentation*. (JetBrains Official Site).

59. Tzutalin. (2015). "LabelImg: An Open-source Annotation Tool." (GitHub).

60. Bhattarai, S. (2018). "Plant Diseases Dataset." (Kaggle).

61. Bochkovskiy, A., et al. (2020). "YOLOv4: Optimal Speed and Accuracy of Object Detection." *arXiv preprint*.

62. Ultralytics. (2023). "YOLOv8 Documentation." (GitHub).

63. Powers, D. M. (2011). "Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation." *Journal of Machine Learning Technologies*.

64. Lipton, Z. C., et al. (2015). "Thresholding Classifiers to Maximize F1 Score." *arXiv preprint*.

65. Davis, J., & Goadrich, M. (2006). "The relationship between Precision-Recall and ROC curves." *ICML*.

66. Saito, T., & Rehmsmeier, M. (2015). "The precision-recall plot is more informative than the ROC plot." *PLOS ONE*

ДОДАТОК А

Програмний код (лістинг) застосунку train_yolo.py

```
import os
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
import seaborn as sns
from ultralytics import YOLO
import yaml
import pandas as pd
import torch
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print(f"Використано пристрій: {device}")

dataset_path = r"
output_dir = os.path.join(dataset_path, 'OUT')

if not os.path.exists(output_dir):
    os.makedirs(output_dir)

class_map = {
    'Apple___Apple_scab': 0,
    'Apple___Black_rot': 1,
    'Apple___Cedar_apple_rust': 2,
    'Apple___healthy': 3,
    'Blueberry___healthy': 4,
    'Cherry_(including_sour)___Powdery_mildew': 5,
    'Cherry_(including_sour)___healthy': 6,
    'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot': 7,
    'Corn_(maize)___Common_rust_': 8,
    'Corn_(maize)___Northern_Leaf_Blight': 9,
    'Corn_(maize)___healthy': 10,
    'Grape___Black_rot': 11,
    'Grape___Esca_(Black_Measles)': 12,
    'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)': 13,
```

```
    'Grape___healthy': 14,
    'Orange___Haunglongbing_(Citrus_greening)': 15,
    'Peach___Bacterial_spot': 16,
    'Peach___healthy': 17,
    'Pepper,_bell___Bacterial_spot': 18,
    'Pepper,_bell___healthy': 19,
    'Potato___Early_blight': 20,
    'Potato___Late_blight': 21,
    'Potato___healthy': 22,
    'Raspberry___healthy': 23,
    'Soybean___healthy': 24,
    'Squash___Powdery_mildew': 25,
    'Strawberry___Leaf_scorch': 26,
    'Strawberry___healthy': 27,
    'Tomato___Bacterial_spot': 28,
    'Tomato___Early_blight': 29,
    'Tomato___Late_blight': 30,
    'Tomato___Leaf_Mold': 31,
    'Tomato___Septoria_leaf_spot': 32,
    'Tomato___Spider_mites Two-spotted_spider_mite': 33,
    'Tomato___Target_Spot': 34,
    'Tomato___Tomato_Yellow_Leaf_Curl_Virus': 35,
    'Tomato___Tomato_mosaic_virus': 36,
    'Tomato___healthy': 37,
}

data_yaml = {
    'train': os.path.join(dataset_path, 'train_yolo'),
    'val': os.path.join(dataset_path, 'val_yolo'),
    'nc': len(class_map),
    'names': list(class_map.keys())
}

yaml_path = os.path.join(output_dir, 'data.yaml')

with open(yaml_path, 'w') as yaml_file:
```

```
    yaml.dump(data_yaml, yaml_file)

MODEL_PATH = r""
#model = YOLO("yolov8s.pt")
model = YOLO(MODEL_PATH)

model.train(
    data=yaml_path,
    epochs=10,
    imgsz=256,
    batch=8,
    project=output_dir,
    name='yolov8_model',
    save=True ,
    patience=5,
    verbose=True,
    augment=False,
    device='CPU',
    save_period=1,
    resume=True,
)
```

ДОДАТОК Б

Програмний код (лістинг) застосунку class_map_yolo.py

```
import os
yolo_dir = r"" #dataset path
def generate_class_map(yolo_dir):
    class_map = {}

    for class_name in sorted(os.listdir(yolo_dir)):
        class_path = os.path.join(yolo_dir, class_name)
        if os.path.isdir(class_path):

            txt_files = [f for f in os.listdir(class_path) if f.endswith('.txt')]
            if not txt_files:
                print(f"Warning: No .txt files found in {class_path}. Skipping.")
                continue

            txt_path = os.path.join(class_path, txt_files[0])
            with open(txt_path, 'r') as f:
                line = f.readline().strip()

            class_id = int(line.split()[0])
            class_map[class_name] = class_id

    return class_map

def main():
    class_map = generate_class_map(yolo_dir)

    print("class_map = {")
    for class_name, class_id in class_map.items():
        print(f"    '{class_name}': {class_id},")
    print("}")

if __name__ == "__main__":
    main()
```

ДОДАТОК В

Програмний код (лістинг) застосунку yolo_test_web.py

```
from flask import Flask, request, render_template_string, redirect, url_for, send_file
import io
import base64
from PIL import Image, ImageOps
from ultralytics import YOLO
import cv2

app = Flask(__name__)

MODEL_PATH = r""
model = YOLO(MODEL_PATH)

HTML_TEMPLATE = """
<!doctype html>
<html lang="en">
<head>
  <title>YOLO Detection</title>
</head>
<body>
  <h1>Upload an image</h1>
  <form method="post" enctype="multipart/form-data">
    <input type="file" name="image" accept="image/*" required>
    <button type="submit">Process</button>
  </form>
  {% if image_url %}
    <h2>Result</h2>
    <img src="{{ image_url }}" alt="Processed image">
    <h3>Statistics:</h3>
    <ul>
    {% for cls, count in stats.items() %}
      <li>{{ cls }}: {{ count }}</li>
    {% endfor %}
    </ul>
```

```
        <br>
        <a href="{{ url_for('download') }}">Download processed image</a>
        <br><br>
        <form method="get" action="{{ url_for('index') }}">
            <button type="submit">Back</button>
        </form>
    {% endif %}
</body>
</html>
"""

processed_image = None
detection_stats = {}

def pad_to_square(image):

    width, height = image.size
    max_side = max(width, height)
    delta_w = max_side - width
    delta_h = max_side - height
    padding = (delta_w // 2, delta_h // 2, delta_w - delta_w // 2, delta_h - delta_h // 2)
    new_image = ImageOps.expand(image, padding, fill=(0, 0, 0))
    return new_image

@app.route("/", methods=["GET", "POST"])
def index():
    global processed_image, detection_stats
    image_url = None

    if request.method == "POST":

        file = request.files["image"]
        input_image = Image.open(file.stream)
        squared_image = pad_to_square(input_image)
        results = model(squared_image)
        processed_image_array = results[0].plot()
```

```
        processed_image_array = cv2.cvtColor(processed_image_array, cv2.COLOR_BGR2RGB)
        processed_image = Image.fromarray(processed_image_array)

        detection_stats = {}
        for box in results[0].boxes:
            cls_name = results[0].names[int(box.cls)]
            detection_stats[cls_name] = detection_stats.get(cls_name, 0) + 1

        buffer = io.BytesIO()
        processed_image.save(buffer, format="JPEG")
        buffer.seek(0)
        image_base64 = base64.b64encode(buffer.getvalue()).decode("utf-8")
        image_url = f"{image_base64}"

    return render_template_string(HTML_TEMPLATE, image_url=image_url, stats=detection_stats)


@app.route("/download")
def download():
    global processed_image

    if processed_image is not None:
        buffer = io.BytesIO()
        processed_image.save(buffer, format="JPEG")
        buffer.seek(0)
        return send_file(buffer, as_attachment=True, download_name="processed_image.jpg", mimetype="image/jpeg")
    else:
        return redirect(url_for("index"))


if __name__ == "__main__":
    app.run(debug=True)
```